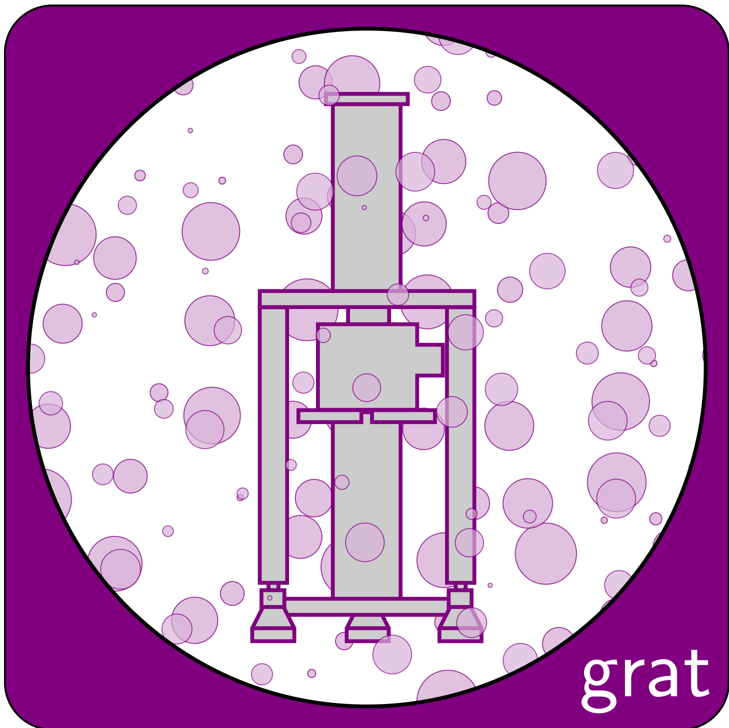

grat

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 . . . 21

8.3 mod_constants::celestial_object_data Type Reference . . . 21

8.3.1 Detailed Description . . . 22

8.4 mod_cmdline::cmd_line_arg Type Reference . . . 22

8.4.1 Detailed Description . . . 22

8.5 mod_date::dateandmjd Type Reference . . . 22

8.5.1 Detailed Description . . . 22

8.6 mod_constants::density_info Type Reference . . . 22

8.6.1 Detailed Description . . . 23

8.7 mod_constants::earth_data Type Reference . . . 23

8.7.1 Detailed Description . . . 23

8.8 mod_constants::earth_density Type Reference . . . 23

8.8.1 Detailed Description 23
8.9 mod_constants::earth_gravity Type Reference 23
8.9.1 Detailed Description 24
8.10 mod_cmdline::field_info Type Reference 24
8.10.1 Detailed Description 24
8.11 mod_data::file Type Reference 24
8.11.1 Detailed Description 25
8.12 mod_constants::gravity_data Type Reference 25
8.12.1 Detailed Description 25
8.13 mod_green::green_common_info Type Reference 25
8.13.1 Detailed Description 25
8.14 mod_green::green_functions Type Reference 25
8.14.1 Detailed Description 26
8.15 mod_cmdline::green_index Type Reference 26
8.15.1 Detailed Description 26
8.16 mod_cmdline::index_info Type Reference 26
8.16.1 Detailed Description 26
8.17 mod_cmdline::info_info Type Reference 26
8.17.1 Detailed Description 27
8.18 mod_data::level_info Type Reference 27
8.18.1 Detailed Description 27
8.19 mod_site::lp_info Type Reference 27
8.19.1 Detailed Description 27
8.20 mod_3d Module Reference 27
8.20.1 Detailed Description 28
8.21 mod_admit Module Reference 28
8.21.1 Detailed Description 28
8.21.2 Member Function/Subroutine Documentation 28
8.21.2.1 parse_admit 28
8.22 mod_aggf Module Reference 28
8.22.1 Detailed Description 29
8.22.2 Member Function/Subroutine Documentation 29
8.22.2.1 aggf 29
8.22.2.2 aggfd 29
8.22.2.3 bouger 30
8.22.2.4 gn_thin_layer 30
8.22.2.5 simple_def 30
8.23 mod_atmosphere Module Reference 31
8.23.1 Detailed Description 31
8.23.2 Member Function/Subroutine Documentation 31

8.23.2.1 geop2geom 31
8.23.2.2 standard_density 31
8.23.2.3 standard_gravity 32
8.23.2.4 standard_pressure 32
8.23.2.5 standard_temperature 32
8.24 mod_cmdline Module Reference 32
8.24.1 Detailed Description 33
8.24.2 Member Function/Subroutine Documentation 33
8.24.2.1 collect_args 33
8.24.2.2 get_command_cleaned 34
8.25 mod_constants Module Reference 34
8.25.1 Detailed Description 35
8.26 mod_data Module Reference 35
8.26.1 Detailed Description 36
8.26.2 Member Function/Subroutine Documentation 36
8.26.2.1 bilinear 36
8.26.2.2 check 37
8.26.2.3 get_dimension 37
8.26.2.4 get_scale_and_offset 37
8.26.2.5 get_value 37
8.26.2.6 nctime2date 37
8.26.2.7 parse_model 38
8.27 mod_date Module Reference 38
8.27.1 Detailed Description 38
8.27.2 Member Function/Subroutine Documentation 38
8.27.2.1 parse_date 38
8.27.2.2 string2date 39
8.28 mod_green Module Reference 39
8.28.1 Detailed Description 40
8.28.2 Member Function/Subroutine Documentation 40
8.28.2.1 convolve 40
8.28.2.2 parse_green 40
8.28.2.3 printmoreverbose 40
8.29 mod_mjd Module Reference 41
8.29.1 Detailed Description 41
8.29.2 Member Function/Subroutine Documentation 41
8.29.2.1 invmjd 41
8.29.2.2 jd 41
8.29.2.3 mjd 42
8.30 mod_normalization Module Reference 42

8.30.1 Detailed Description . . . 42
8.31 mod_parser Module Reference . . . 42
8.31.1 Detailed Description . . . 43
8.31.2 Member Function/Subroutine Documentation . . . 43
8.31.2.1 dataname . . . 43
8.31.2.2 intro . . . 43
8.31.2.3 parse_info . . . 43
8.31.2.4 parse_moreverbose . . . 44
8.31.2.5 print_version . . . 44
8.32 mod_polygon Module Reference . . . 44
8.32.1 Detailed Description . . . 45
8.32.2 Member Function/Subroutine Documentation . . . 45
8.32.2.1 chkgon . . . 45
8.32.2.2 ncross . . . 45
8.32.2.3 parse_polygon . . . 46
8.32.2.4 read_polygon . . . 46
8.33 mod_printing Module Reference . . . 46
8.33.1 Detailed Description . . . 47
8.34 mod_site Module Reference . . . 47
8.34.1 Detailed Description . . . 47
8.34.2 Member Function/Subroutine Documentation . . . 47
8.34.2.1 read_site_file . . . 47
8.35 mod_spherical Module Reference . . . 48
8.35.1 Detailed Description . . . 48
8.35.2 Member Function/Subroutine Documentation . . . 48
8.35.2.1 spher_area . . . 48
8.35.2.2 spher_trig . . . 48
8.35.2.3 spher_trig_inverse . . . 49
8.36 mod_utilities Module Reference . . . 49
8.36.1 Detailed Description . . . 50
8.36.2 Member Function/Subroutine Documentation . . . 50
8.36.2.1 count_records_to_read . . . 50
8.36.2.2 d2r . . . 51
8.36.2.3 file_exists . . . 51
8.36.2.4 is_numeric . . . 51
8.36.2.5 ispline . . . 51
8.36.2.6 ntokens . . . 52
8.36.2.7 r2d . . . 52
8.36.2.8 size_ntimes_denser . . . 52
8.36.2.9 spline . . . 52

8.36.2.10 spline_interpolation 53
8.37 mod_cmdline::model_index Type Reference 53
8.37.1 Detailed Description 53
8.38 mod_site::more_site_heights Type Reference 54
8.38.1 Detailed Description 54
8.39 mod_cmdline::moreverbose_index Type Reference 54
8.39.1 Detailed Description 54
8.40 mod_cmdline::moreverbose_info Type Reference 54
8.40.1 Detailed Description 55
8.41 mod_printing::output_info Type Reference 55
8.41.1 Detailed Description 55
8.42 mod_cmdline::poly_index Type Reference 55
8.42.1 Detailed Description 55
8.43 mod_polygon::polygon_data Type Reference 56
8.43.1 Detailed Description 56
8.44 mod_polygon::polygon_info Type Reference 56
8.44.1 Detailed Description 56
8.45 mod_constants::pressure_data Type Reference 56
8.45.1 Detailed Description 56
8.46 mod_printing::printing_info Type Reference 57
8.46.1 Detailed Description 57
8.47 mod_cmdline::range Type Reference 57
8.47.1 Detailed Description 57
8.48 mod_site::site_info Type Reference 57
8.48.1 Detailed Description 58
8.49 mod_cmdline::subfield_info Type Reference 58
8.49.1 Detailed Description 58
8.50 mod_constants::temperature_data Type Reference 58
8.50.1 Detailed Description 58
8.51 mod_cmdline::transfer_sp_info Type Reference 58
8.51.1 Detailed Description 59
8.52 mod_cmdline::warnings_info Type Reference 59
8.52.1 Detailed Description 59

9 File Documentation 61

9.1 grat/doc/figures/interpolation_ilustration.sh File Reference 61
9.2 interpolation_ilustration.sh 61
9.3 grat/src/grat.f90 File Reference 61
9.4 grat.f90 61
9.5 grat/src/mod_admit.f90 File Reference 64

9.6 mod_admit.f90 65
9.7 grat/src/mod_aggf.f90 File Reference 66
9.7.1 Detailed Description 67
9.8 mod_aggf.f90 67
9.9 grat/src/mod_cmdline.f90 File Reference 70
9.9.1 Detailed Description 71
9.10 mod_cmdline.f90 71
9.11 grat/src/mod_green.f90 File Reference 74
9.12 mod_green.f90 74
9.13 grat/src/mod_normalization.f90 File Reference 85
9.14 mod_normalization.f90 86
9.15 grat/src/mod_polygon.f90 File Reference 86
9.15.1 Detailed Description 86
9.16 mod_polygon.f90 86
9.17 grat/src/real_vs_standard.f90 File Reference 91
9.18 real_vs_standard.f90 91
9.19 grat/src/value_check.f90 File Reference 91
9.19.1 Detailed Description 91
9.20 value_check.f90 92

10 Example Documentation 95

10.1 example_aggf.f90 95
10.2 grat_usage.sh 103

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 unneceserry information whitch are normally collected goes to /dev/null file. This is *nix system default trash. For other system or file system organization, please change this value in mod_cmdline module.

Attention

grat and value_check needs a netCDF library**?**

Copyright

Copyright 2013 by Marcin Rajner
This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

Chapter 2

License

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

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

                            Preamble

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

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

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

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

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

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

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

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

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

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

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

                       TERMS AND CONDITIONS

  0. Definitions.

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

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

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

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

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

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

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

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

  1. Source Code.

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

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

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

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

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

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

  2. Basic Permissions.

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

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

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

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

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

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

  4. Conveying Verbatim Copies.

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

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

  5. Conveying Modified Source Versions.

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

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

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

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

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

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
```

```
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

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

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

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

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

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

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

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

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

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

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

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

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

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

  7. Additional Terms.

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

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

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

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

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

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

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

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

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

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

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

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

  8. Termination.

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

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

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

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

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

  9. Acceptance Not Required for Having Copies.

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

  10. Automatic Licensing of Downstream Recipients.

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

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

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

  11. Patents.

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

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

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

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

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

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

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

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

  12. No Surrender of Others' Freedom.

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

  13. Use with the GNU Affero General Public License.

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

  14. Revised Versions of this License.

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

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

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

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

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

  15. Disclaimer of Warranty.

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

  16. Limitation of Liability.

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

  17. Interpretation of Sections 15 and 16.

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


2.1 Usage

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

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

Chapter 3

External resources

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

Chapter 4

polygon_check

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

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 . . . 21
mod_cmdline::cmd_line_arg . . . 22
mod_date::dateandmjd . . . 22
mod_constants::density_info . . . 22
mod_constants::earth_data . . . 23
mod_constants::earth_density . . . 23
mod_constants::earth_gravity . . . 23
mod_cmdline::field_info . . . 24
mod_data::file . . . 24
mod_constants::gravity_data . . . 25
mod_green::green_common_info . . . 25
mod_green::green_functions . . . 25
mod_cmdline::green_index . . . 26
mod_cmdline::index_info . . . 26
mod_cmdline::info_info . . . 26
mod_data::level_info . . . 27
mod_site::lp_info . . . 27
mod_3d . . . 27
mod_admit . . . 28
mod_aggf . . . 28
mod_atmosphere . . . 31
mod_cmdline . . . 32
mod_constants
Define constant values . . . 34
mod_data
This modele gives routines to read, and write data . . . 35
mod_date . . . 38
mod_green . . . 39
mod_mjd . . . 41
mod_normalization . . . 42
mod_parser . . . 42
mod_polygon . . . 44
mod_printing . . . 46
mod_site . . . 47
mod_spherical . . . 48
mod_utilities . . . 49

mod_cmdline::model_index . . . 53
mod_site::more_site_heights . . . 54
mod_cmdline::moreverbose_index . . . 54
mod_cmdline::moreverbose_info . . . 54
mod_printing::output_info . . . 55
mod_cmdline::poly_index . . . 55
mod_polygon::polygon_data . . . 56
mod_polygon::polygon_info . . . 56
mod_constants::pressure_data . . . 56
mod_printing::printing_info . . . 57
mod_cmdline::range . . . 57
mod_site::site_info . . . 57
mod_cmdline::subfield_info . . . 58
mod_constants::temperature_data . . . 58
mod_cmdline::transfer_sp_info . . . 58
mod_cmdline::warnings_info . . . 59

Chapter 7

File Index

7.1 File List

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

grat/dat/**help.hlp** . . . ??
grat/doc/**LICENSE** . . . ??
grat/doc/figures/interpolation_ilustration.sh . . . 61
grat/doc/figures/**podklad_do_convolution.sh** . . . ??
grat/doc/figures/**polygon_ilustration.sh** . . . ??
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/**barometric_formula.f90** . . . ??
grat/src/grat.f90 . . . 61
grat/src/**mod_3d.f90** . . . ??
grat/src/mod_admit.f90 . . . 65
grat/src/mod_aggf.f90
This module contains utitlities for computing Atmospheric Gravity Green Functions . . . 67
grat/src/**mod_atmosphere.f90** . . . ??
grat/src/mod_cmdline.f90
This module gather cmd line arguments . . . 71
grat/src/**mod_constants.f90** . . . ??
grat/src/**mod_data.f90** . . . ??
grat/src/**mod_date.f90** . . . ??
grat/src/mod_green.f90 . . . 74
grat/src/**mod_mjd.f90** . . . ??
grat/src/mod_normalization.f90 . . . 86
grat/src/**mod_parser.f90** . . . ??
grat/src/mod_polygon.f90
Some routines to deal with inclusion or exclusion of polygons . . . 86
grat/src/**mod_printing.f90** . . . ??
grat/src/**mod_site.f90** . . . ??
grat/src/**mod_spherical.f90** . . . ??
grat/src/**mod_utilities.f90** . . . ??
grat/src/**mytest.sh** . . . ??
grat/src/**polygon_check.f90** . . . ??
grat/src/real_vs_standard.f90 . . . 91
grat/src/value_check.f90 . . . 92

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 121 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 40 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 90 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 25 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 9 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 107 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 65 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 61 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 58 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 20 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 16 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 23 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 96 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 113 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 54 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**
- logical **all** =.false.

8.18.1 Detailed Description

Definition at line 48 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 14 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 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 140 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 272 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 259 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 293 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)
  Compute standard temperature [K] for specific height [km].
- real(dp) function geop2geom (geopotential_height, inverse)
  Compute geometric height from geopotential heights.

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 241 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*)

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 **potential3d** =.false.
- logical **dryrun**
- logical **result_total** =.false.
- logical **result_component** =.true.

8.24.1 Detailed Description

Definition at line 8 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 140 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 199 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** = 8
- integer, parameter **sp** = 4
- real(dp), parameter **r0** = 6356.766
- 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)
- type(atmosphere_data), parameter **atmosphere** = atmosphere_data (pressure = pressure_data (standard = 101325), temperature = temperature_data (standard = 288.15))
- type(earth_data), parameter **earth** = earth_data (mass = 5.97219e24, 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, mass = 7.35e22)
- type(celestial_object_data), parameter **sun** = celestial_object_data (distance = 149600000000, mass = 1.99e30)
- type(density_info), parameter **density** = density_info (water = 1000)

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)

Public Attributes

- type(file), dimension(:),
  allocatable **model**
- type(level_info) **level**

8.26.1 Detailed Description

This modele gives routines to read, and write data.

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

Author

M. Rajner

Date

2013-03-04

Definition at line 12 of file mod_data.f90.

8.26.2 Member Function/Subroutine Documentation

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

Performs bilinear interpolation.

Author

Marcin Rajner

Date

2013-05-07

Definition at line 949 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 781 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 442 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.

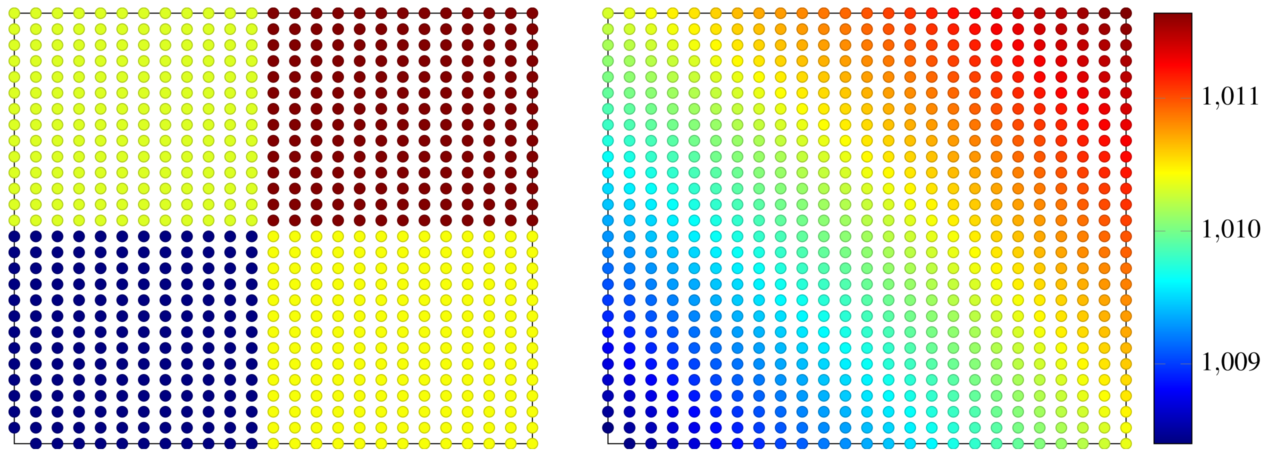
from**?** see `http://www.unidata.ucar.edu/software/netcdf/docs/BestPractices.html`

Definition at line 758 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 819 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 527 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 62 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 22 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 240 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**

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 372 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 40 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 1134 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) *abbreviation*)

Attach full dataname by abbreviation.

Date

2013-03-21

Author

M. Rajner

Definition at line 689 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 220 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 482 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 436 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 592 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

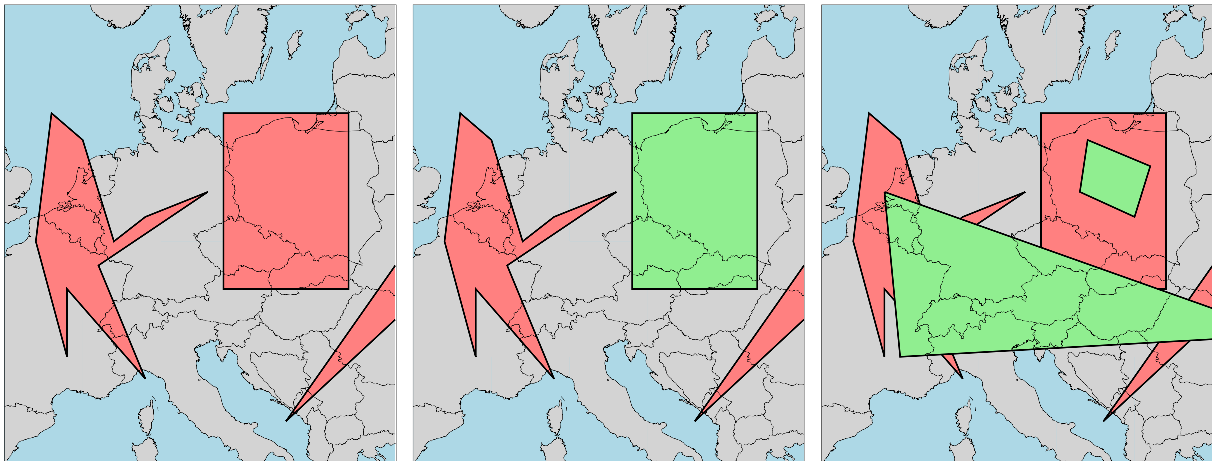


Figure 8.1: capt

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.

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 321 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

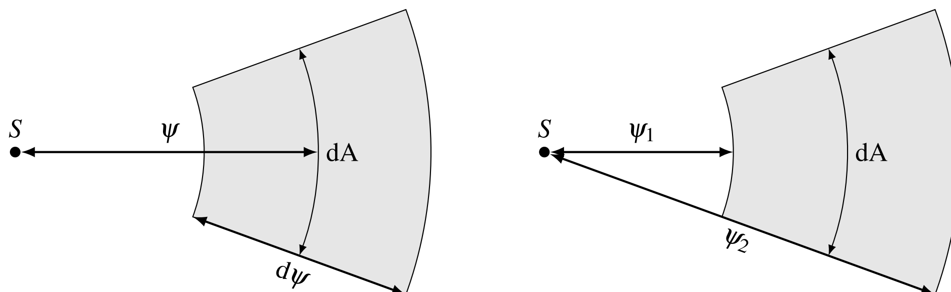
8.35.2.1 real(dp) function mod_spherical::spher_area (real(dp), intent(in) *distance,* real(dp), intent(in) *ddistance,* real(dp), intent(in) *azstp,* real(dp), intent(in), optional *radius,* logical, intent(in), optional *alternative_method*)

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 369 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 341 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 293 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 354 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 404 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) **tp**
- integer(2) **tpf**
- integer(2) **rho**
- integer(2) **vt**

8.37.1 Detailed Description

Definition at line 87 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 93 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 35 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 **nan** = .false.
- character(10) **form** ="en13.3"

8.41.1 Detailed Description

Definition at line 34 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 90 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 34 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 18 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**

8.47.1 Detailed Description

Definition at line 48 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

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 19 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 16 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 72 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 80 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

9.2 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 l
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@l
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**

9.4 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 !! unneceserry 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
00060   ! use omp_lib parallel computation not yet enabled
00061   use mod_parser, only: intro
00062   use mod_data
00063   use mod_date
00064   use mod_green, only: convolve, green
00065   use mod_site, only: print_site_summary, site
00066   use mod_cmdline
00067   use mod_admit, only: admit
00068   use mod_utilities, only: bubble_sort
00069
00070   implicit none
00071   real(dp) :: cpu(2)
00072   integer :: isite, i, idate, start, iprogress = 0
00073   logical :: first_waning = .true.
00074
00075
00076   ! program starts here with time stamp
00077   call cpu_time(cpu(1))
00078
00079   ! gather cmd line option decide where to put output
00080   call intro( &
00081     program_calling = "grat", &
00082       version = "pre-alpha", &
00083       accepted_switches="VSBLGPqoFIDLvhRrMOAHUwJQ", &
00084       cmdlineargs=.true.)
00085
00086     start = 0
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     if(output%header) then
```

```
00099       write (output%unit, '(a8,3(x,a9))', advance="no") "name", "lat", "lon", "h"
00100     endif
00101
00102     if(output%header) then
00103       if (method(1)) then
00104         write (output%unit,'(a13)',advance='no'), "G1D"
00105       endif
00106       if (method(2).or.method(3)) then
00107         if (result_component) then
00108           do i = 1, size(green)
00109             if (green(i)%dataname.eq."GE") then
00110               if (inverted_barometer) then
00111                 write (output%unit,'(a13$)'), trim(green(i)%dataname)//"_IB"
00112               else
00113                 write (output%unit,'(a13$)'), trim(green(i)%dataname)//"_NIB"
00114               endif
00115             else
00116               write (output%unit,'(a13$)'), trim(green(i)%dataname)
00117             endif
00118           enddo
00119           if (inverted_barometer.and.non_inverted_barometer) then
00120             write (output%unit,'(a13$)'), "GE_NIB"
00121           endif
00122         endif
00123         if (result_total) write (output%unit,'(a13)',advance='no'), "G2D"
00124       endif
00125     endif
00126
00127     if(output%header) then
00128       write (output%unit, *)
00129     endif
00130
00131     ! read only once Land-sea, reference surface pressure
00132     if (ind%model%ls.ne.0) then
00133       call get_variable(model(ind%model%ls))
00134     endif
00135     if (ind%model%rsp.ne.0) then
00136       call get_variable(model(ind%model%rsp))
00137     endif
00138     if (ind%model%hrsp.ne.0) then
00139       call get_variable(model(ind%model%hrsp))
00140     endif
00141
00142     if (inverted_landsea_mask.and.ind%model%ls.ne.0) then
00143       model(ind%model%ls)%data = int(abs(model(ind%model%ls)%data-1))
00144     endif
00145
00146
00147     do idate = start, size (date)
00148       if (idate.ge.1) then
00149         if(.not.(output%nan).and.modulo(date(idate)%date(4),6).ne.0) then
00150           if (first_waning) call print_warning  &
00151             ("hours not matching model dates (0,6,12,18) are rejecting and not shown in output")
00152           first_waning=.false.
00153           cycle
00154         endif
00155       endif
00156       do isite = 1, size(site)
00157         iprogress = iprogress + 1
00158
00159         do i = 1, size(model)
00160           if(model(i)%if) then
00161             select case (model(i)%dataname)
00162             case ("SP", "T", "GP", "VT")
00163               if ( &
00164                 .not.(model(i)%autoloadname.eq."ERA" &
00165                 .and.(model(i)%dataname.eq."GP".or.model(i)%dataname.eq."VT")) &
00166                 .and.(idate.eq.1.and. model(i)%autoload &
00167                 .or. (  &
00168                 model(i)%autoload &
00169                 .and. .not. date(idate)%date(1).eq.date(idate-1)%date(1) &
00170                 ) &
00171                 ) &
00172                 ) then
00173                 call model_aliases(model(i), year=date(idate)%date(1))
00174               else if ( &
00175                 idate.eq.1.and. model(i)%autoload &
00176                 .or. (  &
00177                 model(i)%autoload &
00178                 .and. .not.( &
00179                 date(idate)%date(1).eq.date(idate-1)%date(1) &
00180                 .and.date(idate)%date(2).eq.date(idate-1)%date(2) &
00181                 ) &
00182                 ) &
00183                 ) then
00184                 call model_aliases( &
00185                   model(i), year=date(idate)%date(1), month=date(idate)%date(2))
```

```
00186             endif
00187             if (size(date).eq.0.and.model(i)%exist) then
00188               call get_variable(model(i))
00189               elseif (model(i)%exist) then
00190               call get_variable(model(i), date = date(idate)%date)
00191             endif
00192           endselect
00193         endif
00194       enddo
00195       if (any(.not.model%exist).and..not.output%nan) cycle
00196 
00197 
00198       if (level%all.and..not.allocated(level%level)) then
00199         allocate(level%level(size(model(ind%model%gp)%level)))
00200         level%level=model(ind%model%gp)%level
00201       endif
00202 
00203       ! sort levels for 3D method
00204       call bubble_sort(level%level)
00205 
00206       ! if ocean mass should be conserved (-O C)
00207       if (ocean_conserve_mass) then
00208         if (ind%model%sp.ne.0 .and. ind%model%ls.ne.0) then
00209           if(size(date).eq.0) then
00210             call conserve_mass(model(ind%model%sp), model(ind%model%ls), &
00211               inverted_landsea_mask = inverted_landsea_mask)
00212           else
00213             call conserve_mass(model(ind%model%sp), model(ind%model%ls), &
00214               date=date(idate)%date, &
00215               inverted_landsea_mask = inverted_landsea_mask)
00216           endif
00217         endif
00218       endif
00219 
00220       ! calculate total mass if asked for
00221       if (ind%moreverbose%t.ne.0) then
00222         if (size(date).eq.0) then
00223           call total_mass(model(ind%model%sp))
00224         else
00225           call total_mass(model(ind%model%sp), date=date(idate)%date)
00226         endif
00227       endif
00228 
00229 
00230       if (idate.gt.0) then
00231         write(output%unit, '(f12.3,x,i4.4,5(i2.2),x)', advance="no") &
00232           date(idate)%mjd, date(idate)%date
00233       endif
00234       write (output%unit, '(a8,2(x,f9.4),x,f9.3,$)' ), &
00235         site(isite)%name, &
00236         site(isite)%lat,  &
00237         site(isite)%lon,  &
00238         site(isite)%height
00239       if (method(1)) then
00240         write (output%unit, "("// output%form // '$)'), &
00241           admit( &
00242           site(isite), &
00243           date=date(idate)%date &
00244           )
00245       endif
00246 
00247       if (method(2).or.method(3)) then
00248         ! perform convolution
00249         call convolve(site(isite), date = date(idate))
00250       endif
00251 
00252       write(output%unit,*)
00253 
00254       if (output%unit.ne.output_unit.and..not.(quiet.and.quiet_step.eq.0)) then
00255         open(unit=output_unit, carriagecontrol='fortran')
00256         call cpu_time(cpu(2))
00257         call progress(                    &
00258           100*iprogress/(max(size(date),1) &
00259           *max(size(site),1)),             &
00260           cpu(2)-cpu(1), &
00261           every=quiet_step &
00262           )
00263       endif
00264     enddo
00265   enddo
00266 
00267   ! execution time-stamp
00268   call cpu_time(cpu(2))
00269   if (output%unit.ne.output_unit.and..not.(quiet.and.quiet_step.eq.0)) then
00270     call progress(100*iprogress/(max(size(date),1)*max(size(site),1)), cpu(2)-cpu(1), every=1)
00271     close(output_unit)
00272   endif
```

```
00273     write(log%unit, '("Execution time:",1x,f10.4," seconds")') cpu(2)-cpu(1)
00274     if (output%time) write(output%unit, '("Execution time:",1x,f10.4," seconds")') cpu(2)-cpu(1)
00275     write(log%unit, form_separator)
00276   end program
```

9.5 grat/src/mod_admit.f90 File Reference

Data Types

- module mod_admit

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 &
00103         ! .and.ind%model%rsp.ne.0) then
00104           ! call get_value (                   &
00105           ! model=model(ind%model%hrsp),    &
00106           ! lat=site_%lat,                  &
00107           ! lon=site_%lon,                  &
00108           ! val=hrsp,                       &
00109           ! level=1,                        &
00110           ! method = info(1)%interpolation &
00111           ! )
00112
00113           ! rsp = standard_pressure(      &
00114           ! height=site_%height,        &
00115           ! h_zero=hrsp,                &
00116           ! p_zero=rsp,                 &
00117           ! method=transfer_sp%method,  &
00118           ! temperature=t,              &
00119           ! use_standard_temperature    &
00120           ! = ind%model%t.eq.0,         &
00121           ! nan_as_zero=.false.)
00122
00123           ! elseif(ind%model%hrsp.ne.0) then
00124           ! if (first_warning) call print_warning("@RSP not found but @HRSP and -U given")
00125           ! elseif(ind%model%rsp.ne.0) then
00126           ! if (first_warning) call print_warning("@HRSP not found but @RSP and -U given")
00127           ! endif
00128         endif
00129
00130         if (ind%model%rsp.ne.0) val = val-rsp
00131         admit = admitance%value*1.e-2 * val
00132
00133         if (first_warning) first_warning=.false.
00134     end function
00135
00136     ! ==============================================================================
00137     !> \date 2013.10.15
00138     !! \author Marcin Rajner
00139     ! ==============================================================================
00140     subroutine parse_admit(cmd_line_entry)
00141         use mod_cmdline
00142         use mod_printing
00143         type (cmd_line_arg) :: cmd_line_entry
00144         if (cmd_line_entry%field(1)%subfield(1)%name.ne."") then
00145           read(cmd_line_entry%field(1)%subfield(1)%name, *) admitance%value
00146         endif
00147         write(log%unit, '('//form%t2//',a,x,f6.2,x,a)') "admitance:", admitance%value, "uGal/hPa"
00148         ! if (size(cmd_line_entry%field(1)%subfield).gt.1 &
00149         ! .and.cmd_line_entry%field(1)%subfield(2)%name.ne." ") then
00150           ! admitance%level=cmd_line_entry%field(1)%subfield(2)%name
00151           ! else
```

```
00152           ! admitance%level="none"
00153           ! endif
00154           ! write(log%unit, form%i2) "level:", admitance%level
00155       end subroutine
00156   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, zmax, dz,    &
00114     t_zero,            &
00115     h,                 &
00116     first_derivative_h, &
00117     first_derivative_z, &
00118     fels_type,         &
00119     method,            &
00120     predefined,        &
00121     rough)
00122
00123   use mod_constants, only: dp, pi, earth, gravity, atmosphere, r_air
00124   use mod_utilities, only: d2r
00125   use mod_atmosphere
00126   use mod_normalization, only : green_normalization
00127
00128   real(dp), intent(in)           :: psi       ! spherical distance from site [rad]
00129   real(dp), intent(in),optional  :: &
00130     zmin,    & ! minimum height, starting point [m]      (default = 0)
00131     zmax,    & ! maximum height, ending point   [m]      (default = 60000)
00132     dz,      & ! integration step               [m]      (default = 0.1 -> 10 cm)
00133     t_zero,  & ! temperature at the surface     [K]      (default = 15°C i.e., 288.15=t0)
00134     h          ! station height                 [m]      (default = 0)
00135   logical, intent(in), optional :: &
```

```
00136       first_derivative_h, first_derivative_z, predefined, rough
00137     character (len=*), intent(in), optional  :: fels_type, method
00138     character (len=20) :: old_method
00139     real(dp) :: aggf
00140     real(dp) :: zmin_, zmax_, dz_, h_
00141     real(dp) :: j_aux
00142     real(dp) :: rho, l, deltat
00143
00144     real(dp), dimension(:), allocatable, save :: heights, pressures
00145     integer :: i
00146
00147     zmin_ = 0.
00148     zmax_ = 60000.
00149     dz_   = 0.1
00150     h_    = 0.
00151
00152     aggf=0.
00153
00154     if (present(zmin)) zmin_ = zmin
00155     if (present(zmax)) zmax_ = zmax
00156     if (present(  dz))   dz_ = dz
00157     if (present(   h))    h_ = h
00158     if (present(t_zero)) deltat=t_zero
00159
00160     if(allocated(heights)) then
00161       if ( &
00162         ((zmin_ +dz_/2).ne.heights(1)) &
00163         .or.abs((zmax_-dz_/2)-heights(size(heights))).gt.zmax_/1e6 &
00164         .or.nint((zmax_-zmin_)/dz_).ne.size(heights) &
00165         .or.  (present(predefined)) &
00166         .or. method.ne.old_method &
00167         .or. present(t_zero) &
00168         ) then
00169         deallocate(heights)
00170         deallocate(pressures)
00171       endif
00172     endif
00173
00174     if (.not.allocated(heights))  then
00175       allocate(heights(nint((zmax_-zmin_)/dz_)))
00176       allocate(pressures(size(heights)))
00177       do i = 1, size(heights)
00178         heights(i) = zmin_ &
00179           + dz_/2  &
00180           + (i-1) * dz_
00181       enddo
00182
00183       if (present(rough).and.rough) then
00184         ! do not use rough! it is only for testing
00185         do i = 1, size(heights)
00186           pressures(i) = standard_pressure( &
00187             heights(i),                      &
00188             method=method,                   &
00189             dz=dz,                           &
00190             use_standard_temperature=.true.  &
00191             )
00192         enddo
00193       else
00194         pressures(1) = standard_pressure(     &
00195           heights(1),                         &
00196           method = method,                    &
00197           h_zero = zmin_,                     &
00198           dz = dz,                            &
00199           fels_type=fels_type,                &
00200           use_standard_temperature=.true.,    &
00201           temperature = standard_temperature( &
00202           zmin_, fels_type=fels_type)+deltat  &
00203           )
00204         do i = 2, size(heights)
00205           pressures(i) = standard_pressure(                     &
00206             heights(i),                                         &
00207             p_zero = pressures(i-1),                            &
00208             h_zero = heights(i-1),                              &
00209             method = method,                                    &
00210             dz = dz,                                            &
00211             fels_type=fels_type,                                &
00212             use_standard_temperature=.true.,                    &
00213             temperature = standard_temperature(heights(i-1),   &
00214             fels_type=fels_type)+deltat                         &
00215             )
00216         enddo
00217       endif
00218     endif
00219     old_method=method
00220
00221     do i = 1, size(heights)
00222       l = ((earth%radius + heights(i))**2 + (earth%radius + h_)**2 &
```

```
00223       - 2.*(earth%radius + h_)*(earth%radius+heights(i))*cos(psi))**(0.5)
00224     rho = pressures(i)/ r_air / (deltat+standard_temperature(heights(i), fels_type=fels_type))
00225     if (present(first_derivative_h) .and. first_derivative_h) then
00226       ! first derivative (respective to station height)
00227       ! micro Gal height / m
00228       ! see equation 22, 23 in \cite Huang05
00229       j_aux =  ((earth%radius + heights(i) )**2)*(1.-3.*((cos(psi))**2)) -2.*(earth%radius + h_)**2  &
00230         + 4.*(earth%radius+h_)*(earth%radius+heights(i))*cos(psi)
00231       aggf =  aggf + rho * (  j_aux  /  l**5  ) * dz_
00232
00233     else if (present(first_derivative_z) .and. first_derivative_z) then
00234       ! first derivative (respective to column height)
00235       ! according to equation 26 in \cite Huang05
00236       ! micro Gal / hPa / m
00237       if (i.gt.1) exit
00238       aggf = rho *( ((earth%radius + heights(i))*cos(psi)-(earth%radius + h_)) / (l**3))
00239     else
00240       ! GN microGal/hPa
00241       aggf = aggf &
00242         -rho*((earth%radius +heights(i))*cos(psi) - (earth%radius + h_)) / (l**3.)  * dz_
00243     endif
00244   enddo
00245   aggf = aggf/atmosphere%pressure%standard*gravity%constant*green_normalization("m", psi=psi)
00246 end function
00247
00248 ! ============================================================================
00249 !> Compute AGGF GN for thin layer
00250 !!
00251 !! Simple function added to provide complete module
00252 !! but this should not be used for atmosphere layer
00253 !! See eq p. 491 in \cite Merriam92
00254 !! \author M. Rajner
00255 !! \date 2013-03-19
00256 !! \warning psi in radian
00257 !! \todo explanaition ??
00258 ! ============================================================================
00259 function gn_thin_layer (psi)
00260   use mod_constants, only: dp
00261   real(dp), intent(in) :: psi
00262   real(dp) :: gn_thin_layer
00263
00264   gn_thin_layer = 1.627 * psi / sin( psi / 2. )
00265 end function
00266
00267
00268 ! ============================================================================
00269 !> \brief Bouger plate computation
00270 !!
00271 ! ============================================================================
00272 real(dp) function bouger (h, R )
00273   use mod_constants, only: dp, gravity, pi
00274   real(dp), intent(in), optional :: r !< height of point above the cylinder
00275   real(dp), intent(in) ::  h
00276
00277   if (present( r ) ) then
00278     bouger = h + r - sqrt(r**2+h**2)
00279   else
00280     bouger = h
00281   endif
00282   bouger = 2 * pi * gravity%constant * bouger
00283   return
00284 end function
00285
00286 ! ============================================================================
00287 !> Bouger plate computation
00288 !!
00289 !! see eq. page 288 \cite Warburton77
00290 !! \date 2013-03-18
00291 !! \author M. Rajner
00292 ! ============================================================================
00293 function simple_def (R)
00294   use mod_constants, only: dp, earth
00295   real(dp) :: r, delta
00296   real(dp) :: simple_def
00297
00298   delta = 0.22e-11 * r
00299   simple_def = earth%gravity%mean / earth%radius *1000 * &
00300     delta * ( 2. - 3./2. * earth%density%crust / earth%density%mean &
00301     -3./4. * earth%density%crust / earth%density%mean * sqrt(2* (1. )) &
00302     ) * 1000
00303 end function
00304
00305 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 ! ============================================================================
00002 !> \file
00003 !! \brief This module gather cmd line arguments
00004 !!
00005 !! it allows to specify commands with or without spaces therefore it is
00006 !! convienient to use with auto completion of names
00007 ! ============================================================================
00008 module mod_cmdline
00009   use mod_constants, only: dp
00010 
00011   implicit none
00012 
00013   !----------------------------------------------------
00014   ! command line entry
00015   !----------------------------------------------------
00016   type subfield_info
00017     character (len=100) :: name
00018     character (len=100) :: dataname
00019   end type
00020   type field_info
00021     character (len=355) :: full
00022     type(subfield_info), allocatable, &
00023         dimension(:) :: subfield
00024   end type
00025   type cmd_line_arg
00026     character(2) :: switch
00027     type (field_info), allocatable, &
00028         dimension(:) :: field
00029     character (len=455) :: full
00030   end type
00031   type(cmd_line_arg), allocatable, dimension(:) :: cmd_line
00032 
00033   private :: check_if_switch_or_minus
00034 
00035   type moreverbose_info
```

```
00036     character(60) :: name
00037     character(30):: dataname
00038     logical  :: sparse=.false.
00039     logical :: first_call = .true.
00040     integer :: unit
00041     logical :: noclobber = .false.
00042   end type
00043   type(moreverbose_info), allocatable, dimension(:) :: moreverbose
00044
00045   !----------------------------------------------------
00046   ! info
00047   !----------------------------------------------------
00048   type range
00049     real(dp):: start
00050     real(dp):: stop
00051     real(dp):: step
00052     integer :: denser
00053   end type
00054   type info_info
00055     type (range):: distance,azimuth, height
00056     character (1) :: interpolation
00057   end type
00058   type(info_info), dimension(:), allocatable:: info
00059
00060   !----------------------------------------------------
00061   ! general settings
00062   !----------------------------------------------------
00063   logical :: &
00064       inverted_barometer     = .true.  , &
00065       non_inverted_barometer = .false. , &
00066       ocean_conserve_mass    = .false. , &
00067       inverted_landsea_mask  = .false. , &
00068       optimize               = .false. , &
00069       quiet                  = .false.
00070   integer :: quiet_step=50
00071
00072       type transfer_sp_info
00073         logical :: if = .false.
00074         ! by default with 2D method pressure is transfered
00075         ! on topography (@H)
00076         character(20) :: method="standard"
00077       end type
00078       type(transfer_sp_info) transfer_sp
00079
00080       type warnings_info
00081         logical :: if = .true., &
00082           strict=.false., &
00083           time=.false.
00084       end type
00085       type(warnings_info) warnings
00086
00087       type model_index
00088         integer(2) :: sp, t, rsp, ewt, h, ls, hp, hrsp, gp, tp, tpf, rho, vt
00089       end type
00090       type poly_index
00091         integer(2) :: e, n
00092       end type
00093       type moreverbose_index
00094         integer(2) :: p, g, t, a, d, l, n, r, s, o, b, j, v
00095       end type
00096       type green_index
00097         integer(2) :: &
00098           gn          = 0,  & ! green newtonian   - with SP  in Pa
00099           ge          = 0,  & ! green elastic     - with SP  in Pa
00100           gegdt       = 0,  & ! green elastic     - first derivative of gravity part respect to temp (see
      Guo et al., 2004)
00101           gr          = 0,  & ! green radial      - with EWT in mm
00102           ghn         = 0,  & ! green horizontal  - with EWT in mm
00103           ghe         = 0,  & ! green horizontal  - with EWT in mm
00104           gg          = 0,  & ! green gravimetric - with SP  in Pa
00105           ! (like elastic but uses green not normalized according to Merriam)
00106         gndt        = 0,  & ! first derivative respect to temperature
00107           gndh        = 0,  & ! first derivative respect to station height
00108           gndz        = 0,  & ! first derivative respect to column height
00109           gndz2       = 0,  & ! second derivative respect to column height
00110           gnc         = 0,  & ! compute aggf every time
00111           g3d
00112       end type
00113       type index_info
00114         type (model_index)       :: model
00115         type (moreverbose_index) :: moreverbose
00116         type (green_index) :: green
00117         type (poly_index) :: polygon
00118       end type
00119       type(index_info) :: ind
00120
00121       type admitance_info
```

```
00122         logical :: if
00123         real(dp):: value = -0.3
00124       end type
00125       type(admitance_info) :: admitance
00126
00127       logical :: method(3)
00128       logical :: potential3d=.false.
00129       logical :: dryrun
00130
00131       logical :: result_total=.false., result_component=.true.
00132   contains
00133     ! ============================================================================
00134     !> This routine collect command line arguments to one matrix depending on
00135     !! given switches and separators
00136     !!
00137     !! \date 2013.05.21
00138     !! \author Marcin Rajner
00139     ! ============================================================================
00140     subroutine collect_args (dummy)
00141         use mod_utilities, only: ntokens, count_separator
00142         character(*) :: dummy
00143         character(455) :: dummy_aux, dummy_aux2
00144         integer :: i, j, n
00145         integer :: indeks_space, indeks_comma, indeks_at, indeks_colon
00146
00147         allocate(cmd_line(ntokens(dummy)))
00148         do i=1, ntokens(dummy)
00149           indeks_space = index(dummy," ")
00150           cmd_line(i)%full= dummy(1:indeks_space-1)
00151           cmd_line(i)%switch=cmd_line(i)%full(1:2)
00152           allocate(cmd_line(i)%field (count_separator(cmd_line(i)%full,",") + 1))
00153
00154           dummy_aux = cmd_line(i)%full(3:)
00155           do j=1,size(cmd_line(i)%field)
00156             indeks_comma=index(dummy_aux,",")
00157             if (indeks_comma.gt.0) then
00158               cmd_line(i)%field(j)%full=dummy_aux(1:indeks_comma-1)
00159             else
00160               cmd_line(i)%field(j)%full=dummy_aux
00161             endif
00162
00163             allocate(cmd_line(i)%field(j)%subfield &
00164                 (count_separator(cmd_line(i)%field(j)%full,":") + 1))
00165             dummy_aux2 = cmd_line(i)%field(j)%full
00166             do n = 1, count_separator(cmd_line(i)%field(j)%full,":")+1
00167               indeks_colon=index(dummy_aux2,":")
00168               if (indeks_colon.gt.0) then
00169                 cmd_line(i)%field(j)%subfield(n)%name=dummy_aux2(1:indeks_colon-1)
00170               else
00171                 cmd_line(i)%field(j)%subfield(n)%name=dummy_aux2
00172               endif
00173               dummy_aux2=dummy_aux2(indeks_colon+1:)
00174               indeks_at=index(cmd_line(i)%field(j)%subfield(n)%name,"@")
00175               if (indeks_at.gt.0) then
00176                 cmd_line(i)%field(j)%subfield(n)%dataname = &
00177                     cmd_line(i)%field(j)%subfield(n)%name(indeks_at+1:)
00178                 cmd_line(i)%field(j)%subfield(n)%name = &
00179                     cmd_line(i)%field(j)%subfield(n)%name(1:indeks_at-1)
00180               else
00181                 cmd_line(i)%field(j)%subfield(n)%dataname = " "
00182               endif
00183             enddo
00184             dummy_aux=dummy_aux(indeks_comma+1:)
00185           enddo
00186           dummy= dummy(indeks_space+1:)
00187         enddo
00188     end subroutine
00189
00190     ! =============================================================================
00191     !> This subroutine removes unnecesary blank spaces from cmdline entry
00192     !!
00193     !! Marcin Rajner
00194     !! \date 2013-05-13
00195     !! allows specification like '-F file' and '-Ffile'
00196     !! but  if -[0,9] it is treated as number belonging to switch (-S -2)
00197     !! but  if -[\s,:] do not start next command line option
00198     ! =============================================================================
00199     subroutine get_command_cleaned(dummy)
00200         character(*), intent(out) :: dummy
00201         character(355) :: a, b, arg
00202         integer :: i
00203         dummy=" "
00204         do i = 1, iargc()
00205           call get_command_argument(i,a)
00206           call get_command_argument(i+1,b)
00207           if (check_if_switch_or_minus(a)) then
00208             arg = trim(a)
```

```
00209           else
00210             arg=trim(arg)//trim(a)
00211           endif
00212           if(check_if_switch_or_minus(b).or.i.eq.iargc()) then
00213             if(trim(dummy).eq."") then
00214               dummy=trim(arg)
00215             else
00216               dummy=trim(dummy)//" "//trim(arg)
00217             endif
00218           endif
00219         enddo
00220     end subroutine
00221
00222     ! ==============================================================================
00223     !> Check if - starts new option in command line or is just a minus in command
00224     !! line entry
00225     !!
00226     !! if after '-' is space or number or ',' or ':' (field separators) do not start
00227     !! next option for command line
00228     !! If switch return .true. otherwise return .false
00229     !!
00230     !! \author M. Rajner
00231     !! \date 2013-03-19
00232     ! ==============================================================================
00233     function check_if_switch_or_minus(dummy)
00234         use mod_utilities, only: is_numeric
00235         logical:: check_if_switch_or_minus
00236         character(*) :: dummy
00237
00238         check_if_switch_or_minus = .false.
00239         if (dummy(1:1).eq."-") check_if_switch_or_minus = .true.
00240         if (dummy(2:2).eq." ") check_if_switch_or_minus = .false.
00241         if (dummy(2:2).eq.",") check_if_switch_or_minus = .false.
00242         if (dummy(2:2).eq.":") check_if_switch_or_minus = .false.
00243         if (is_numeric(dummy(2:2))) check_if_switch_or_minus = .false.
00244     end function
00245
00246 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

9.12 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 contains
00032 ! ============================================================================
00033 !> This subroutine parse -G option -- Greens function.
00034 !!
00035 !! This subroutines takes the -G argument specified as follows:
00036 !!   -G
00037 !! \author M. Rajner
00038 !! \date 2013-03-06
00039 ! ============================================================================
00040 subroutine parse_green (cmd_line_entry)
00041   use mod_utilities, only: file_exists, is_numeric
00042   use mod_cmdline
00043   use mod_printing
00044   type (cmd_line_arg), optional  :: cmd_line_entry
00045   integer :: i, ii
00046
00047   if (allocated(green)) then
00048     call print_warning("repeated")
00049     return
00050   endif
00051
00052   if (method(3)) then
00053     if (present(cmd_line_entry)) then
00054       allocate (green(size(cmd_line_entry%field)+1))
00055     else
00056       allocate (green(1))
00057     endif
00058     ind%green%g3d=ubound(green,1)
00059     green(ind%green%g3d)%name="merriam"
00060     green(ind%green%g3d)%column=[1, 2]
00061     green(ind%green%g3d)%dataname="G3D"
00062     call read_green(green(ind%green%g3d))
00063   else
00064     allocate (green(size(cmd_line_entry%field)))
00065   endif
00066
00067   if (present(cmd_line_entry)) then
00068     do i = 1, size(cmd_line_entry%field)
00069       write(log%unit, form%i2) trim(basename(trim(cmd_line_entry%field(i)%full)))
00070       green(i)%name = cmd_line_entry%field(i)%subfield(1)%name
00071       if (i.gt.1.and.cmd_line_entry%field(i)%subfield(1)%name.eq."") then
00072         green(i)%name = green(i-1)%name
00073       endif
00074       if (any(green%dataname.eq.cmd_line_entry%field(i)%subfield(1)%dataname )) then
00075         call print_warning("repeated dataname for Green")
00076         continue
00077       else
00078         green(i)%dataname = cmd_line_entry%field(i)%subfield(1)%dataname
00079       endif
00080       do ii=1, 2
00081         green(i)%column(ii) =green(i-1)%column(ii)
00082         green(i)%columndataname(ii) = green(i-1)%columndataname(ii)
00083         if(is_numeric(cmd_line_entry%field(i)%subfield(ii+1)%name ) ) then
00084           read(cmd_line_entry%field(i)%subfield(ii+1)%name, *) green(i)%column(ii)
00085           green(i)%columndataname(ii) = cmd_line_entry%field(i)%subfield(ii+1)%dataname
00086         endif
00087       enddo
00088       call read_green(green(i))
00089     enddo
00090   endif
00091
00092   ! check completness
00093   ! if ( &
00094   ! ! any(green%name.eq."/home/mrajner/src/grat/dat/merriam_green.dat" &
00095   ! ! .and. green%dataname.eq."GNdz" ) &
00096   ! ! .neqv. &
00097   ! any(green%name.eq."/home/mrajner/src/grat/dat/merriam_green.dat" &
00098   ! .and. green%dataname.eq."GNdz2" ) &
00099   ! ) call print_warning("-G: merriam@GNdz should go with merriam @GNdz2")
00100 end subroutine
00101
00102 ! ============================================================================
00103 !> This subroutine read  green file
00104 ! ============================================================================
00105 subroutine read_green (green, print)
00106   use mod_utilities, only: file_exists, skip_header, r2d, d2r
00107   use iso_fortran_env
00108   use mod_printing
00109   use mod_constants, only: earth, pi
00110   use mod_normalization, only: green_normalization
00111
00112   integer :: lines, fileunit, io_status, i
00113   real (dp), allocatable, dimension(:) :: tmp
00114   type(green_functions) :: green
```

```
00115   logical, optional :: print
00116
00117   ! change the paths accordingly
00118   if (.not.file_exists(green%name) &
00119     .and. (.not. green%name.eq."merriam" &
00120     .and.  .not. green%name.eq."huang" &
00121     .and.  .not. green%name.eq."rajner" )) then
00122     green%name="merriam"
00123   endif
00124   select case (green%name)
00125   case ("merriam", "compute", "/home/mrajner/src/grat/dat/merriam_green.dat")
00126     green%name="/home/mrajner/src/grat/dat/merriam_green.dat"
00127     select case (green%dataname)
00128     case("GN")
00129       green%column=[1, 2]
00130     case("GNdt")
00131       green%column=[1, 3]
00132     case("GNdz")
00133       green%column=[1, 4]
00134     case("GNdz2")
00135       green%column=[1, 5]
00136     case("GE")
00137       green%column=[1, 6]
00138     case("GNc")
00139       green%column=[1, 2]
00140     case("G3D")
00141       green%column=[1, 2]
00142     case default
00143       call print_warning( &
00144         "green type not found", &
00145         more=trim(green%dataname), &
00146         error=.true.)
00147     endselect
00148   case ("huang", "/home/mrajner/src/grat/dat/huang_green.dat" )
00149     green%name="/home/mrajner/src/grat/dat/huang_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("GNdh")
00156       green%column=[1, 4]
00157     case("GNdz")
00158       green%column=[1, 5]
00159     case default
00160       call print_warning( &
00161         trim(green%dataname) //" not found in " &
00162         // trim(green%name), error=.true.)
00163     endselect
00164   case ("rajner", "/home/mrajner/src/grat/dat/rajner_green.dat")
00165     green%name="/home/mrajner/src/grat/dat/rajner_green.dat"
00166     select case (green%dataname)
00167     case("GN")
00168       green%column=[1, 2]
00169     case("GNdt")
00170       green%column=[1, 3]
00171     case("GNdh")
00172       green%column=[1, 4]
00173     case("GNdz")
00174       green%column=[1, 5]
00175     case default
00176       call print_warning( &
00177         trim(green%dataname) //" not found in " &
00178         // trim(green%name), error=.true.)
00179       call print_warning(green%dataname //"not found in " // green%name, &
00180         error=.true.)
00181     endselect
00182   endselect
00183
00184   if(green%column(1).ne.0 .and. green%column(2).ne.0) then
00185     allocate(tmp(max(green%column(1), green%column(2))))
00186     lines = 0
00187     open (newunit =fileunit, file=green%name, action="read", status="old")
00188     do
00189       call skip_header(fileunit)
00190       read (fileunit, *, iostat = io_status) tmp
00191       if (io_status == iostat_end) exit
00192       lines = lines + 1
00193     enddo
00194
00195     allocate (green%distance(lines))
00196     allocate (green%data(lines))
00197     rewind(fileunit)
00198     lines = 0
00199     do
00200       call skip_header(fileunit)
00201       lines = lines + 1
```

```
00202       read (fileunit, *, iostat = io_status) tmp
00203       if (io_status == iostat_end) then
00204         close(fileunit)
00205         exit
00206       endif
00207       green%distance(lines) = tmp(green%column(1))
00208       green%data(lines)     = tmp(green%column(2))
00209     enddo
00210     deallocate(tmp)
00211   endif
00212
00213   ! file specific
00214   if (green%name.eq."/home/mrajner/src/grat/dat/merriam_green.dat") then
00215     select case(green%dataname)
00216     case("GNdz")
00217       green%data = green%data * 1.e-3
00218     endselect
00219   endif
00220
00221   if (.not.present(print)) then
00222     write(log%unit, form%i3) trim(basename(trim(green%name))), trim(green%dataname), &
00223       "columns:", green%column, &
00224       "lines:", size(green%distance)
00225   endif
00226
00227   if (green%columndataname(1).eq."R") then
00228     green%distance=(/ (r2d(green%distance(i)), i=1, size(green%distance)) /)
00229     write(log%unit, form_63) "conversion: radians --> to degrees"
00230   endif
00231   if (green%columndataname(2).eq."a2f") then
00232     green%data=green%data  / (earth%radius)*1e12 * earth%gravity%mean
00233     write(log%unit, form_63) "conversion: aplo --> to farrell"
00234   endif
00235   if (green%columndataname(2).eq."f2m") then
00236     green%data= &
00237       -green%data * green_normalization("f2m")
00238     write(log%unit, form_63) "conversion: farrell --> to merriam"
00239   endif
00240 end subroutine
00241
00242 ! ============================================================================
00243 !> Unification:
00244 ! ============================================================================
00245 subroutine green_unification ()
00246   use mod_utilities, only: size_ntimes_denser, spline_interpolation, d2r
00247   use mod_cmdline,   only: info, moreverbose, ind
00248   use mod_printing
00249   use mod_site, only: site
00250
00251   type(green_functions) :: tmpgreen
00252   integer :: i, iinfo, imin, imax, j, ii
00253   integer, allocatable, dimension(:):: which_green, tmp
00254
00255   allocate (green_common(size(info)))
00256   allocate (which_green(size(info)))
00257   allocate (tmp(size(green)))
00258   do iinfo=1, size(info)
00259     if (info(iinfo)%distance%step.eq.0) then
00260       do i = 1, size(green)
00261         tmp(i) = count(                                          &
00262           green(i)%distance.le.info(iinfo)%distance%stop       &
00263           .and.green(i)%distance.ge.info(iinfo)%distance%start &
00264           )
00265       enddo
00266       which_green(iinfo) = maxloc(tmp, 1)
00267
00268       imin=minloc( &
00269         abs(green(which_green(iinfo))%distance - info(iinfo)%distance%start), 1)-1
00270       imax=minloc( &
00271         abs(green(which_green(iinfo))%distance - info(iinfo)%distance%stop), 1)+1
00272
00273       if (imin.lt.1) imin = 1
00274       if (imax.gt.size(green(which_green(iinfo))%distance)) then
00275         imax = size(green(which_green(iinfo))%distance)
00276       endif
00277
00278       allocate(tmpgreen%distance(                                  &
00279         size_ntimes_denser(imax-imin+1, info(iinfo)%distance%denser) &
00280         ))
00281       do ii = 1, imax - imin
00282         do j = 1, info(iinfo)%distance%denser
00283           tmpgreen%distance((ii-1)*info(iinfo)%distance%denser+j) = &
00284             green(which_green(iinfo))%distance(imin+ii-1)          &
00285             +(j-1)*(green(which_green(iinfo))%distance(imin+ii)    &
00286             -green(which_green(iinfo))%distance(imin+ii-1))        &
00287             /info(iinfo)%distance%denser
00288         enddo
```

```
00289       enddo
00290
00291       tmpgreen%distance(size(tmpgreen%distance)) = &
00292         green(which_green(iinfo))%distance(imax)
00293
00294       imin = count(tmpgreen%distance.le.info(iinfo)%distance%start)
00295       imax = size(tmpgreen%distance) - &
00296         count(tmpgreen%distance.ge.info(iinfo)%distance%stop ) + 1
00297
00298       allocate(green_common(iinfo)%distance(imax-imin+1))
00299       green_common(iinfo)%distance =       &
00300         tmpgreen%distance(imin:imax)
00301       green_common(iinfo)%distance(1) =    &
00302         (3/4.*info(iinfo)%distance%start+  &
00303         green_common(iinfo)%distance(2)/4)
00304       green_common(iinfo)%distance(size(green_common(iinfo)%distance)) =      &
00305         (3/4.*info(iinfo)%distance%stop+                                      &
00306         green_common(iinfo)%distance(size(green_common(iinfo)%distance)-1)/4)
00307
00308       allocate(green_common(iinfo)%start(size(green_common(iinfo)%distance)))
00309       allocate(green_common(iinfo)%stop(size(green_common(iinfo)%distance)))
00310
00311       green_common(iinfo)%start=(green_common(iinfo)%distance)
00312       do i =1, size(green_common(iinfo)%distance)
00313         green_common(iinfo)%start(i)=(green_common(iinfo)%distance(i) + &
00314           green_common(iinfo)%distance(i-1) ) / 2.
00315         green_common(iinfo)%stop(i)=(green_common(iinfo)%distance(i) + &
00316           green_common(iinfo)%distance(i+1) ) / 2.
00317       enddo
00318
00319       green_common(iinfo)%start(1)= info(iinfo)%distance%start
00320       green_common(iinfo)%stop(size(green_common(iinfo)%stop)) = &
00321         info(iinfo)%distance%stop
00322       deallocate(tmpgreen%distance)
00323     else
00324       allocate(green_common(iinfo)%distance( &
00325         ceiling( &
00326         (info(iinfo)%distance%stop - info(iinfo)%distance%start) &
00327         /info(iinfo)%distance%step) &
00328         ))
00329       allocate(green_common(iinfo)%start(size(green_common(iinfo)%distance)))
00330       allocate(green_common(iinfo)%stop(size(green_common(iinfo)%distance)))
00331
00332       green_common(iinfo)%start = &
00333         [(info(iinfo)%distance%start + &
00334         (i-1)*info(iinfo)%distance%step, &
00335         i=1, size(green_common(iinfo)%distance)) ]
00336       green_common(iinfo)%stop = green_common(iinfo)%start(2:)
00337       green_common(iinfo)%stop(ubound(green_common(iinfo)%stop)) = info(iinfo)%distance%stop
00338       green_common(iinfo)%distance = &
00339         (green_common(iinfo)%stop + green_common(iinfo)%start)/2
00340     endif
00341
00342     allocate(green_common(iinfo)%data(size(green_common(iinfo)%distance), size(green)))
00343     allocate(green_common(iinfo)%dataname(size(green)))
00344
00345     do i = 1,  size(green_common(iinfo)%data, 2)
00346       call  spline_interpolation(          &
00347         green(i)%distance,                 &
00348         green(i)%data,                     &
00349         size(green(i)%distance),           &
00350         green_common(iinfo)%distance,      &
00351         green_common(iinfo)%data(:, i),     &
00352         size(green_common(iinfo)%distance) &
00353         )
00354       where( &
00355           green_common(iinfo)%distance.gt.green(i)%distance(size(green(i)%distance)) &
00356           .or.green_common(iinfo)%distance.lt.green(i)%distance(1) &
00357           )
00358         green_common(iinfo)%data(:, i)=0
00359       end where
00360       green_common(iinfo)%dataname(i) = green(i)%dataname
00361     enddo
00362   enddo
00363 end subroutine
00364
00365
00366 ! ============================================================================
00367 !> Perform convolution
00368 !!
00369 !! \date 2013-03-15
00370 !! \author M. Rajner
00371 ! ============================================================================
00372 subroutine convolve(site, date)
00373   use mod_constants
00374   use iso_fortran_env
00375   use mod_site, only : site_info
```

```
00376   use mod_cmdline
00377   use mod_utilities, &
00378     only: d2r, r2d, datanameunit, mmwater2pascal, countsubstring
00379   use mod_spherical
00380   use mod_data
00381   use mod_date, only : dateandmjd
00382   use mod_polygon
00383   use mod_printing
00384   use mod_normalization, only: green_normalization
00385   use mod_aggf, only: aggf
00386   use mod_atmosphere, only: standard_pressure, standard_temperature
00387   use mod_3d
00388   type(site_info), intent(in) :: site
00389   type(dateandmjd), intent(in), optional :: date
00390
00391   integer  :: igreen, idist, iazimuth, nazimuth
00392   real(dp) :: azimuth, dazimuth
00393   real(dp) :: lat, lon, height, area, tot_area, tot_area_used
00394   real(dp) :: val(size(model)), old_val_sp, old_val_rsp
00395   integer  :: i, j, npoints, iheight, nheight
00396   integer(2) :: iok(size(polygon))
00397
00398   real(dp) :: normalize, aux
00399   real(dp), allocatable, dimension(:) :: azimuths, &
00400       heights, pressures, temperatures
00401   logical :: header_p = .true.
00402
00403   real(dp) :: h1,h2, v1,v2, p_int !temporary
00404   real(dp) :: rsp
00405
00406
00407   if (transfer_sp%if) then
00408     if (ind%model%hp.eq.0) call print_warning("no @HP with -U", error=.true.)
00409     if (ind%model%h .eq.0) call print_warning("no @H  with -U", error=.true.)
00410   endif
00411
00412   if(.not.allocated(green_common)) then
00413     call green_unification()
00414   endif
00415
00416   if (site%lp%if) then
00417     val=0
00418     do i=1, size(site%lp%date)
00419       if(all(site%lp%date(i, 1:6).eq.date%date(1:6))) then
00420         val=site%lp%data(i)
00421         exit
00422       endif
00423       if(i.eq.size(site%lp%date)) &
00424         ! call print_warning("date not found in @LP")
00425       val=sqrt(-1.)
00426     enddo
00427   endif
00428
00429   if (.not. allocated(result)) then
00430     if (any(green%dataname.eq."GE").and.inverted_barometer &
00431       .and. non_inverted_barometer) then
00432       allocate(result(size(green)+1))
00433     else
00434       allocate(result(size(green)))
00435     endif
00436   endif
00437
00438   npoints       = 0
00439   area          = 0
00440   tot_area      = 0
00441   tot_area_used = 0
00442
00443   result=0
00444   rsp=0
00445
00446   if (ind%green%gnc.ne.0) close(output_unit)
00447
00448   do igreen = 1, size(green_common)
00449     do idist = 1, size(green_common(igreen)%distance)
00450       if (allocated(azimuths)) deallocate (azimuths)
00451       if (info(igreen)%azimuth%step.eq.0) then
00452         nazimuth = &
00453           (info(igreen)%azimuth%stop-info(igreen)%azimuth%start)/360 * &
00454           max(int(360*sin(d2r(green_common(igreen)%distance(idist)))), 100) * &
00455           info(igreen)%azimuth%denser
00456         if (nazimuth.eq.0) nazimuth=1
00457         dazimuth= (info(igreen)%azimuth%stop-info(igreen)%azimuth%start)/nazimuth
00458       else
00459         dazimuth = info(igreen)%azimuth%step
00460         nazimuth= (info(igreen)%azimuth%stop-info(igreen)%azimuth%start)/dazimuth
00461       endif
00462
```

```
00463
00464       ! calculate area using spherical formulae
00465       area = spher_area(                          &
00466         d2r(green_common(igreen)%start(idist)), &
00467         d2r(green_common(igreen)%stop(idist)),  &
00468         d2r(dazimuth),                          &
00469         radius=earth%radius,                    &
00470         alternative_method=.true.)
00471
00472       ! normalization according to Merriam (1992)
00473       normalize= 1e8 / &
00474         (green_normalization("m", psi = d2r(green_common(igreen)%distance(idist))))
00475
00476       allocate(azimuths(nazimuth))
00477       azimuths = [(info(igreen)%azimuth%start + (i-1) * dazimuth, i= 1, nazimuth)]
00478
00479       do iazimuth  = 1, nazimuth
00480         azimuth = azimuths(iazimuth)
00481
00482         npoints = npoints + 1
00483         tot_area=tot_area+area
00484
00485         ! get lat and lon of point
00486         call spher_trig &
00487           (d2r(site%lat), d2r(site%lon), &
00488           d2r(green_common(igreen)%distance(idist)), d2r(azimuth), lat, lon, domain=.true.)
00489
00490         ! read polygons
00491         if (ind%polygon%e.ne.0 .or. ind%polygon%n.ne.0) then
00492           do i =1, size(polygon)
00493             if (polygon(i)%if) then
00494               call chkgon(r2d(lon), r2d(lat), polygon(i), iok(i))
00495             endif
00496           enddo
00497         endif
00498
00499         ! get LS
00500         if (ind%model%ls.ne.0.and.inverted_barometer) then
00501           call get_value( &
00502             model(ind%model%ls), r2d(lat), r2d(lon), val(ind%model%ls), &
00503             level=1, method=info(igreen)%interpolation, date=date%date)
00504         endif
00505
00506         if (iok(1).eq.1 & .and. int(val(ind%model%ls)).eq.1) then
00507           tot_area_used = tot_area_used +area
00508         endif
00509
00510         ! GE, GN, ...
00511         if (any([&
00512             ind%green%gn,   ind%green%ge, ind%green%gg, &
00513             ind%green%gndt, ind%green%gnc, ind%green%gegdt, ind%green%g3d &
00514             ].ne.0) &
00515           ) then
00516
00517           if ( &
00518             ind%model%sp.ne.0.and.(model(ind%model%sp)%if &
00519             .or.model(ind%model%sp)%if_constant_value) &
00520             ) then
00521
00522             ! get SP
00523             if (.not.(site%lp%if.and.green_common(igreen)%distance(idist).lt.0.3)) then
00524               call get_value(                                           &
00525                 model(ind%model%sp), r2d(lat), r2d(lon), val(ind%model%sp), &
00526                 level=1, method = info(igreen)%interpolation, date=date%date)
00527             endif
00528             old_val_sp=val(ind%model%sp)
00529
00530             if (.not.isnan(val(ind%model%sp))) then
00531
00532               ! get RSP if given
00533               if (ind%model%rsp.ne.0) then
00534                 call get_value(                                              &
00535                   model(ind%model%rsp), r2d(lat), r2d(lon), val(ind%model%rsp), &
00536                   level=1, method = info(igreen)%interpolation)
00537               endif
00538               old_val_rsp=val(ind%model%rsp)
00539
00540               if(transfer_sp%if.and..not.all([ind%model%rsp, ind%model%hrsp].ne.0)) then
00541                 call print_warning("@RSP or @HRSP with -U is missing", error=.true.)
00542               else
00543                 call get_value( &
00544                   model(ind%model%hrsp), r2d(lat), r2d(lon), val(ind%model%hrsp), &
00545                   level=1, method = info(igreen)%interpolation)
00546               endif
00547
00548
00549               ! get T
```

```
00550                   if (ind%model%t.ne.0 &
00551                     .and.( &
00552                     transfer_sp%if &
00553                     .or.any(([ &
00554                     ind%green%gndt, &
00555                     ind%green%gegdt &
00556                     ]).ne.0) &
00557                     ) &
00558                     ) then
00559                     call get_value( &
00560                       model(ind%model%t), r2d(lat), r2d(lon), val(ind%model%t), &
00561                       level=1, method=info(igreen)%interpolation, date=date%date)
00562                   endif
00563 
00564                   ! get HP
00565                   if (ind%model%hp.ne.0 &
00566                     .and.( &
00567                     transfer_sp%if &
00568                     .or. ind%green%g3d.ne.0 &
00569                     ) &
00570                     ) then
00571                     call get_value( &
00572                       model(ind%model%hp), r2d(lat), r2d(lon), val(ind%model%hp), &
00573                       level=1, method = info(igreen)%interpolation)
00574                   endif
00575 
00576                   ! get H
00577                   if (ind%model%h.ne.0 &
00578                     .and.( &
00579                     transfer_sp%if &
00580                     .or.any(([ &
00581                     ind%green%gndt, ind%green%gndz, ind%green%gndz2, ind%green%gndh, &
00582                     ind%green%g3d &
00583                     ]).ne.0) &
00584                     ) &
00585                     ) then
00586                     if (optimize.and.green_common(igreen)%distance(idist).gt.20) then
00587                       val(ind%model%h)=val(ind%model%hp)
00588                     else
00589                       call get_value( &
00590                         model(ind%model%h), r2d(lat), r2d(lon), val(ind%model%h), &
00591                         level=1, method = info(igreen)%interpolation)
00592                     endif
00593                   endif
00594 
00595                   if (ind%model%sp.ne.0) then
00596                     ! transfer SP if necessary on terrain
00597                     if (transfer_sp%if &
00598                       .and.any([ &
00599                       ind%green%ge, &
00600                       ind%green%gnc, &
00601                       ind%green%g3d, &
00602                       ind%green%gegdt, &
00603                       ind%green%gg &
00604                       ].ne.0) &
00605                       ) then
00606                       val(ind%model%sp) = standard_pressure( &
00607                         height=val(ind%model%h),           &
00608                         h_zero=val(ind%model%hp),          &
00609                         p_zero=old_val_sp,          &
00610                         method=transfer_sp%method,         &
00611                         temperature=val(ind%model%t),      &
00612                         use_standard_temperature           &
00613                         = ind%model%t.eq.0,                &
00614                         nan_as_zero=.false.)
00615 
00616                       if(all([ind%model%rsp, ind%model%hrsp].ne.0)) then
00617                         val(ind%model%rsp) = standard_pressure(          &
00618                           height=val(ind%model%h),                       &
00619                           h_zero=val(ind%model%hrsp),                    &
00620                           p_zero=old_val_rsp,                            &
00621                           method=transfer_sp%method,                     &
00622                           temperature=val(ind%model%t),                  &
00623                           use_standard_temperature                       &
00624                           = ind%model%t.eq.0,                            &
00625                           nan_as_zero=.false.)
00626                       endif
00627                     endif
00628 
00629                     if(ind%model%rsp.ne.0) val(ind%model%sp) = val(ind%model%sp) - val(ind%model%rsp)
00630 
00631                     ! if the cell is not over sea and inverted barometer assumption was not set
00632                     ! and is not excluded by polygon
00633                     if ((ind%polygon%e.ne.0.and.iok(ind%polygon%e).ne.0).or.(ind%polygon%e.eq.0)) then
00634                       !IB or NIB
00635                       if (.not.(ind%model%ls.ne.0.and.inverted_barometer.and.int(val(ind%model%ls)).eq.0)) then
00636                         ! GE
```

```
00637                   if (ind%green%ge.ne.0) then
00638                     result(ind%green%ge) = result(ind%green%ge) +        &
00639                       val(ind%model%sp) *                                &
00640                       green_common(igreen)%data(idist, ind%green%ge) * &
00641                       area * normalize
00642                   endif
00643
00644                   ! GEGdt pressure part from Guo 2004
00645                   if (ind%green%gegdt.ne.0) then
00646                     result(ind%green%gegdt) = result(ind%green%gegdt) +     &
00647                       val(ind%model%sp) *                                   &
00648                       val(ind%model%t) * 1e-4 *                             &
00649                       green_common(igreen)%data(idist, ind%green%gegdt) * &
00650                       area * normalize
00651                   endif
00652
00653                   ! GG
00654                   if (ind%green%gg.ne.0) then
00655                     aux = mmwater2pascal(val(ind%model%sp), inverted=.true.) &
00656                       * area/ (d2r(green_common(igreen)%distance(idist)) * &
00657                       earth%radius*1e18)
00658                     result(ind%green%gg) = result(ind%green%gg) +           &
00659                       green_common(igreen)%data(idist, ind%green%gg) *     &
00660                       aux * 1e8 ! m s-2 -> microGal
00661                   endif
00662                 endif
00663
00664                 ! ! GE NIB if both IB and NIB wanted
00665                 if (inverted_barometer.and.non_inverted_barometer) then
00666                   if (ind%green%ge.ne.0) then
00667                     result(ubound(result)) = result(ubound(result)) +        &
00668                       val(ind%model%sp) *                                &
00669                       green_common(igreen)%data(idist, ind%green%ge) * &
00670                       area * normalize
00671                   endif
00672                 endif
00673               endif
00674
00675               if (                                                   &
00676                 (ind%polygon%n.ne.0.and.iok(ind%polygon%n).ne.0)      &
00677                 .or.(ind%polygon%n.eq.0)                              &
00678                 ) then
00679
00680                 !3D
00681                 if (method(3)) then
00682                   ! print *
00683                   if (ind%model%rsp.eq.0) then
00684                     call print_warning("3D but no RSP")
00685                   endif
00686
00687                   if (allocated(heights)) deallocate(heights)
00688                   if (allocated(pressures)) deallocate(pressures)
00689                   if (allocated(temperatures)) deallocate(temperatures)
00690
00691                   nheight= &
00692                       ceiling((info(igreen)%height%stop &
00693                       -max(info(igreen)%height%start,val(ind%model%h))) &
00694                       /info(igreen)%height%step)
00695
00696                   allocate(heights(nheight))
00697                   allocate(pressures(nheight))
00698                   allocate(temperatures(nheight))
00699
00700                   do iheight=1, nheight
00701                     heights(iheight)=max(info(igreen)%height%start, val(ind%model%h)) &
00702                         +(iheight-0.5)*info(igreen)%height%step
00703                   enddo
00704
00705                   if (.not. allocated(level%height)) allocate (level%height(size(level%level)))
00706                   if (.not. allocated(level%temperature)) allocate (level%temperature(size(level%level)))
00707
00708                   do i=1,size(level%level)
00709                     call get_value(                                                         &
00710                         model(ind%model%gp), r2d(lat), r2d(lon), level%height(i),            &
00711                         level=level%level(i), method = info(igreen)%interpolation, date=date%date)
00712                     if (ind%model%vt.ne.0) then
00713                       call get_value(                                                       &
00714                           model(ind%model%vt), r2d(lat), r2d(lon), level%temperature(i),
      &
00715                           level=level%level(i), method = info(igreen)%interpolation, date=date%date)
00716                     endif
00717                   enddo
00718
00719                   i=1
00720                   do while(level%height(i).lt.heights(1).and.i.ne.size(level%level))
00721                     i=i+1
00722                   end do
```

```
00723                     do iheight=1, nheight
00724
00725                       if (iheight.eq.1) then
00726                         h1=val(ind%model%h)
00727                         v1=val(ind%model%sp)+val(ind%model%rsp)
00728                         h2=level%height(i)
00729                         v2=1.e2*dble(level%level(i))
00730
00731                         temperatures(iheight)=level%temperature(i)-6.5e-3*(val(ind%model%h)-val(ind%model
      %hp))
00732
00733                         pressures(iheight) = standard_pressure(        &
00734                           heights(iheight),                            &
00735                           p_zero=val(ind%model%sp)+val(ind%model%rsp), &
00736                           h_zero=val(ind%model%h), &
00737                           method="standard",                           &
00738                           use_standard_temperature=.true.,             &
00739                           temperature=val(ind%model%t)                 &
00740                           )
00741
00742                       else
00743                         do while(level%height(i+1).lt.heights(iheight).and. i.ne.size(level%level))
00744                           i=i+1
00745                         end do
00746
00747                         ! temperature linear interpolation
00748                         if(i.lt.size(level%level)) then
00749                           temperatures(iheight)= &
00750                             level%temperature(i) &
00751                             + (level%temperature(i+1)-level%temperature(i)) &
00752                             /(level%height(i+1)-level%height(i))*(heights(iheight)-level%height(i))
00753                         else
00754                           temperatures(iheight)= &
00755                             level%temperature(i)
00756                         endif
00757
00758                         if(heights(iheight-1).lt.level%height(i).and.(heights(iheight).gt.level%height(i)))
       then
00759                           h1=level%height(i)
00760                           v1=1.e2*dble(level%level(i))
00761                           h2=level%height(i+1)
00762                           v2=1.e2*dble(level%level(i+1))
00763
00764                           pressures(iheight)= standard_pressure( &
00765                             height=heights(iheight), &
00766                             p_zero=1.e2*dble(level%level(i)), &
00767                             h_zero=level%height(i), &
00768                             method="standard", &
00769                             use_standard_temperature=.true., &
00770                             temperature=temperatures(iheight), &
00771                             nan_as_zero=.true.)
00772                         else
00773                           pressures(iheight)= standard_pressure( &
00774                             height=heights(iheight), &
00775                             p_zero=pressures(iheight-1), &
00776                             h_zero=heights(iheight-1), &
00777                             method="standard", &
00778                             use_standard_temperature=.true., &
00779                             temperature=temperatures(iheight), &
00780                             nan_as_zero=.true.)
00781                         endif
00782                       endif
00783
00784                       ! if (i.lt.size(level%level)) then
00785                         ! p_int=exp(dlog(v1) +(dlog(v2)-dlog(v1))*(heights(iheight)-h1)/(h2-h1))
00786                         ! if (p_int.gt.1e29)   p_int=0
00787                         ! pressures(iheight)=p_int
00788                       ! endif
00789
00790
00791                       if (potential3d) then
00792                         result(ind%green%g3d) = result(ind%green%g3d)      &
00793                           + potential(                                     &
00794                           psi1=d2r(green_common(igreen)%start(idist)),     &
00795                           psi2=d2r(green_common(igreen)%stop(idist)),      &
00796                           dazimuth=d2r(dazimuth),                          &
00797                           h=site%height,                                   &
00798                           z1= heights(iheight)-info(igreen)%height%step/2, &
00799                           z2= heights(iheight)+info(igreen)%height%step/2  &
00800                           )                                                &
00801                           * pressures(iheight)/(temperatures(iheight))
00802                       else
00803                         result(ind%green%g3d) = result(ind%green%g3d) &
00804                           + geometry(psi=d2r(green_common(igreen)%distance(idist)), h=site%height, z=
      heights(iheight)) &
00805                           * pressures(iheight)/(temperatures(iheight))   &
00806                           * area * info(igreen)%height%step
```

```
00807                 endif
00808               enddo
00809             endif
00810             ! if
00811             ! stop
00812
00813             !C before GN GNdt etc because it needs SP on H not on site
00814             if(ind%green%gnc.ne.0) then
00815               if ( &
00816                 any([ &
00817                 ind%model%sp, &
00818                 ind%model%hp, &
00819                 ind%model%h, &
00820                 ind%model%t &
00821                 ].eq.0)) &
00822                 call print_warning("with @GNc you need to give @T @HP @H", error=.true.)
00823               result(ind%green%gnc) = result(ind%green%gnc)  &
00824                 + val(ind%model%sp)                          &
00825                 * aggf(                                      &
00826                 d2r(green_common(igreen)%distance(idist)),   &
00827                 zmin=val(ind%model%h),                       &
00828                 t_zero=val(ind%model%t),                     &
00829                 h=site%height,                               &
00830                 dz =  &
00831                 merge(10._dp, &
00832                 merge(0.1_dp,1._dp, &
00833                 green_common(igreen)%distance(idist).le.1e-5_dp ), &
00834                 green_common(igreen)%distance(idist).ge.1e-2_dp ), &
00835                 method="standard",                           &
00836                 predefined=.true.)                           &
00837                 * area * normalize
00838
00839               if (.not.quiet) then
00840                 open(unit=output_unit, carriagecontrol='fortran')
00841                 call progress(                                               &
00842                   100*igreen*idist                                         &
00843                   /(size(green_common(igreen)%distance)*size(green_common)), &
00844                   every=1 &
00845                   )
00846               endif
00847             endif
00848
00849             ! transfer SP if necessary on site level
00850             if (transfer_sp%if &
00851               .and.any([ &
00852               ind%green%gn, &
00853               ind%green%gndt, &
00854               ind%green%gndz, &
00855               ind%green%gndz2, &
00856               ind%green%gndh &
00857               ].ne.0) &
00858               ) then
00859               val(ind%model%sp) = standard_pressure( &
00860                 height=site%height,               &
00861                 h_zero=val(ind%model%hp),         &
00862                 p_zero=old_val_sp,                &
00863                 method=transfer_sp%method,        &
00864                 temperature=val(ind%model%t),     &
00865                 use_standard_temperature          &
00866                 = ind%model%t.eq.0,               &
00867                 nan_as_zero=.false.)
00868
00869               if(all([ind%model%rsp, ind%model%hrsp].ne.0)) then
00870                 val(ind%model%rsp) = standard_pressure(          &
00871                   height=site%height,                            &
00872                   h_zero=val(ind%model%hrsp),                    &
00873                   p_zero=old_val_rsp,                            &
00874                   method=transfer_sp%method,                     &
00875                   temperature=val(ind%model%t),                  &
00876                   use_standard_temperature                       &
00877                   = ind%model%t.eq.0,                            &
00878                   nan_as_zero=.false.)
00879               endif
00880               if(ind%model%rsp.ne.0) val(ind%model%sp) = val(ind%model%sp) - val(ind%model%rsp)
00881             endif
00882
00883             ! GN
00884             if (ind%green%gn.ne.0) then
00885               result(ind%green%gn) = result(ind%green%gn) +      &
00886                 val(ind%model%sp) *                              &
00887                 green_common(igreen)%data(idist, ind%green%gn) * &
00888                 area * normalize
00889             endif
00890
00891             ! reference for 3D method
00892             if (ind%green%g3d.ne.0) then
00893               rsp = rsp + &
```

```
00894                     val(ind%model%rsp) *                              &
00895                     green_common(igreen)%data(idist, ind%green%gn) * &
00896                     area * normalize
00897                 endif
00898 
00899                 ! GNdt
00900                 if (ind%green%gndt.ne.0) then
00901                   if (any(                                                &
00902                     [ind%model%sp, ind%model%t, ind%model%rsp           &
00903                     ].eq.0)) &
00904                     call print_warning("not enough data model for GNdt", &
00905                     error=.true.)
00906                   result(ind%green%gndt) = result(ind%green%gndt) +       &
00907                     val(ind%model%sp)                                   &
00908                     * green_common(igreen)%data(idist, ind%green%gndt)    &
00909                     * (val(ind%model%t)-atmosphere%temperature%standard) &
00910                     *  area * normalize
00911                 endif
00912 
00913                 ! GNdh
00914                 if (ind%green%gndh.ne.0) then
00915                   if (any(                                                &
00916                     [ &
00917                     ind%model%sp, ind%model%h, ind%model%rsp           &
00918                     ].eq.0)) &
00919                     call print_warning("not enough data model for GNdh", &
00920                     error=.true.)
00921                   result(ind%green%gndh) = result(ind%green%gndh) +       &
00922                     val(ind%model%sp)                                   &
00923                     * green_common(igreen)%data(idist, ind%green%gndh)    &
00924                     * (val(ind%model%h)-site%height) &
00925                     *  area * normalize
00926                 endif
00927 
00928                 ! GNdz
00929                 if (ind%green%gndz.ne.0) then
00930                   if (any(                                                &
00931                     [ &
00932                     ind%model%sp, ind%model%h, ind%model%rsp           &
00933                     ].eq.0)) &
00934                     call print_warning("not enough data model for GNdz", &
00935                     error=.true.)
00936                   result(ind%green%gndz) = result(ind%green%gndz) +       &
00937                     val(ind%model%sp)                                   &
00938                     * green_common(igreen)%data(idist, ind%green%gndz)    &
00939                     * (val(ind%model%h)-site%height) &
00940                     *  area * normalize
00941                 endif
00942 
00943 
00944                 ! GNdz2
00945                 if (ind%green%gndz2.ne.0) then
00946                   if (any(                                                &
00947                     [ &
00948                     ind%model%sp, ind%model%h, ind%model%rsp           &
00949                     ].eq.0)) &
00950                     call print_warning("not enough data model for GNdz2", &
00951                     error=.true.)
00952                   result(ind%green%gndz2) = result(ind%green%gndz2) +                 &
00953                     val(ind%model%sp)                                             &
00954                     * green_common(igreen)%data(idist, ind%green%gndz2)             &
00955                     * ( (val(ind%model%h)-site%height)                             &
00956                     /(earth%radius * d2r(green_common(igreen)%distance(idist))) )**2 &
00957                     *  area * normalize
00958                 endif
00959               endif
00960             endif
00961           else
00962             result=sqrt(-1.)
00963           endif
00964         else
00965           call print_warning("@SP is required with -M2D -G", error=.true.)
00966         endif
00967       endif
00968 
00969       ! surface loads from EWT
00970       if (                                                                          &
00971         ind%green%gr.ne.0                                                         &
00972         .or.ind%green%ghn.ne.0                                                    &
00973         .or.ind%green%ghe.ne.0                                                    &
00974         ) then
00975         if ((ind%polygon%e.ne.0.and.iok(ind%polygon%e).ne.0).or.(ind%polygon%e.eq.0)) then
00976           if (.not.(ind%model%ls.ne.0.and.inverted_barometer.and.int(val(ind%model%ls)).eq.0)) then
00977             call get_value(                                                         &
00978               model(ind%model%ewt), r2d(lat), r2d(lon), val(ind%model%ewt),        &
00979               level=1, method = info(igreen)%interpolation, date=date%date)
00980             aux = (val(ind%model%ewt))   *                                          &
```

```
00981               area/d2r(green_common(igreen)%distance(idist)) *                                    &
00982               1./earth%radius/1e12* 1e3 ! m -> mm
00983             if (isnan(aux)) aux = 0
00984             if (ind%green%gr.ne.0) then
00985               result(ind%green%gr) = result(ind%green%gr) +        &
00986                 green_common(igreen)%data(idist, ind%green%gr) &
00987                 * aux
00988
00989               if (ind%green%ghn.ne.0) then
00990                 result(ind%green%ghn) = result(ind%green%ghn) +       &
00991                   green_common(igreen)%data(idist, ind%green%ghn) * &
00992                   aux * (- cos(d2r(azimuth)))
00993               endif
00994               if (ind%green%ghe.ne.0) then
00995                 result(ind%green%ghe) = result(ind%green%ghe) +       &
00996                   green_common(igreen)%data(idist, ind%green%ghe) * &
00997                   aux * (- sin(d2r(azimuth)))
00998               endif
00999             endif
01000           endif
01001         endif
01002       endif
01003
01004       ! moreverbose point: -L@p
01005       if(ind%moreverbose%p.ne.0) then
01006         if (header_p.and. output%header) then
01007           if(size(green_common).gt.1) &
01008             write(moreverbose(ind%moreverbose%p)%unit, "(a2, x$)") "i"
01009
01010           write(moreverbose(ind%moreverbose%p)%unit, &
01011             '(a8, 8a13, $)')                          &
01012             "name", "lat", "lon",                   &
01013             "distance", "azimuth",                  &
01014             "lat", "lon",                           &
01015             "area", "totarea"
01016
01017           if (result_component) then
01018             write(moreverbose(ind%moreverbose%p)%unit, &
01019               '(a13, $)')                 &
01020               (trim(green(i)%dataname), &
01021               i=lbound(green, 1),       &
01022               ubound(green, 1)          &
01023               )
01024           endif
01025
01026           if (result_total) then
01027             write(moreverbose(ind%moreverbose%p)%unit, &
01028               '(a13, $)') "total"
01029           endif
01030           if (.not.moreverbose(ind%moreverbose%p)%sparse) then
01031             write(moreverbose(ind%moreverbose%p)%unit,                              &
01032               '(<size(model)>a12)', advance='no' )                                &
01033               (trim(model(i)%dataname), i=lbound(model, 1), ubound(model, 1))
01034           endif
01035           if (size(iok).gt.0) then
01036             write(moreverbose(ind%moreverbose%p)%unit, &
01037               '(<size(iok)>(a3, i1))'),                &
01038               ("ok", i, i =1, ubound(iok, 1))
01039           else
01040             write(moreverbose(ind%moreverbose%p)%unit, *)
01041           endif
01042           header_p=.false.
01043         endif
01044         if (                                                   &
01045           .not.moreverbose(ind%moreverbose%p)%sparse     &
01046           .or.                                           &
01047           (moreverbose(ind%moreverbose%p)%sparse         &
01048           .and.(azimuth==azimuths(ubound(azimuths, 1))) &
01049           )                                              &
01050           ) then
01051
01052           if(size(green_common).gt.1) &
01053             write(moreverbose(ind%moreverbose%p)%unit, "(i2, x$)") igreen
01054           write(moreverbose(ind%moreverbose%p)%unit,           &
01055             '(a8, 6' // output%form //',2 en13.3, $)'),        &
01056             site%name, site%lat, site%lon,                 &
01057             green_common(igreen)%distance(idist), azimuth, &
01058             r2d(lat), r2d(lon), area, tot_area
01059           if (result_component)                            &
01060             write(moreverbose(ind%moreverbose%p)%unit, &
01061             '(' // output%form //'$)'),                &
01062             (result(i), i =1, size(result))
01063           if (result_total) &
01064             write(moreverbose(ind%moreverbose%p)%unit, &
01065             '(' // output%form //'$)'), sum(result(1:size(green)))
01066           if (.not.moreverbose(ind%moreverbose%p)%sparse) then
01067             do i=1, size(val)
```

```
01068                 call get_value(                            &
01069                   model(i), r2d(lat), r2d(lon), val(i), &
01070                   level=1,                                 &
01071                   method = info(igreen)%interpolation,   &
01072                   date=date%date)
01073               enddo
01074               write(moreverbose(ind%moreverbose%p)%unit, &
01075                 '(<size(model)>en12.2, $)') val
01076             endif
01077             if (size(iok).gt.0) then
01078               write(moreverbose(ind%moreverbose%p)%unit, &
01079                 '(<size(iok)>(i4))'), iok
01080             else
01081               write(moreverbose(ind%moreverbose%p)%unit, * )
01082             endif
01083           endif
01084         endif
01085
01086         ! moreverbose auxilary to draw: -L@a
01087         if(ind%moreverbose%a.ne.0) then
01088           call printmoreverbose(                                    &
01089             d2r(site%lat), d2r(site%lon), d2r(azimuth), d2r(dazimuth), &
01090             d2r(green_common(igreen)%start(idist)),                 &
01091             d2r(green_common(igreen)%stop(idist))                   &
01092             )
01093         endif
01094       enddo
01095     enddo
01096   enddo
01097
01098   if (ind%green%g3d.ne.0) then
01099     result(ind%green%g3d)=-result(ind%green%g3d)*gravity%constant*1e8/r_air-rsp
01100   endif
01101
01102   ! results to output
01103   if (result_component) write (output%unit, "(" // output%form // '$)') result
01104   if (result_total)     write (output%unit, "(" // output%form // '$)') sum(result(1:size(green)))
01105
01106   ! summary: -L@s
01107   if (ind%moreverbose%s.ne.0) then
01108     if (output%header) write(moreverbose(ind%moreverbose%s)%unit, '(2a8, 3a12)' ) &
01109       "station", "npoints", "area", "area/R2", "t_area_used"
01110     write(moreverbose(ind%moreverbose%s)%unit, '(a8, i8, 3en12.2)') &
01111       site%name, npoints, tot_area, tot_area/earth%radius**2, tot_area_used
01112   endif
01113
01114   ! green values : -L@g
01115   if(ind%moreverbose%g.ne.0) then
01116     do i = 1, size(green_common)
01117       do j=1,size(green_common(i)%distance)
01118         write(moreverbose(ind%moreverbose%g)%unit, '(i3,f14.6, 100f14.7)'), &
01119           j, green_common(i)%distance(j), &
01120           green_common(i)%start(j), &
01121           green_common(i)%stop(j), &
01122           green_common(i)%data(j,:), &
01123           green_common(i)%distance(j)-green_common(i)%distance(j-1)
01124       enddo
01125     enddo
01126   endif
01127 end subroutine
01128
01129 ! ============================================================================
01130 !> returns lat and lon of spherical trapezoid
01131 !! \date 2013.07.03
01132 !! \author Marcin Rajner
01133 ! ============================================================================
01134 subroutine printmoreverbose (latin, lonin, azimuth, azstep, distancestart, distancestop)
01135   use mod_spherical, only : spher_trig
01136   use mod_cmdline,   only : moreverbose, ind
01137   use mod_utilities, only : r2d
01138
01139   real(dp), intent(in) :: azimuth, azstep, latin, lonin
01140   real(dp) ::  lat, lon, distancestart, distancestop
01141
01142   call spher_trig(latin, lonin, distancestart, azimuth - azstep/2, lat, lon)
01143   write(moreverbose(ind%moreverbose%a)%unit, '(8f12.6)'), r2d(lat), r2d(lon)
01144   call spher_trig(latin, lonin, distancestop, azimuth - azstep/2, lat, lon)
01145   write(moreverbose(ind%moreverbose%a)%unit, '(8f12.6)'), r2d(lat), r2d(lon)
01146   call spher_trig(latin, lonin, distancestop, azimuth + azstep/2, lat, lon)
01147   write(moreverbose(ind%moreverbose%a)%unit, '(8f12.6)'), r2d(lat), r2d(lon)
01148   call spher_trig(latin, lonin, distancestart, azimuth + azstep/2, lat, lon)
01149   write(moreverbose(ind%moreverbose%a)%unit, '(8f12.6)'), r2d(lat), r2d(lon)
01150   write(moreverbose(ind%moreverbose%a)%unit, '(">")')
01151 end subroutine
01152
01153 ! ============================================================================
01154 !! \date 2013-07-02
```

```
01155 !! \author M. Rajner
01156 !! \warning input spherical distance in radian
01157 !!
01158 !! method:
01159 !!   default see equation in Rajnerdr
01160 !!   spotl    see \cite spotl manual
01161 !!    olssson see \cite olsson2009
01162 !! ===============================================================================
01163 function green_newtonian (psi, h, z, method)
01164   use mod_constants, only: earth, gravity
01165   use mod_normalization, only: green_normalization
01166   real(dp) :: green_newtonian
01167   real(dp), intent (in) :: psi
01168   real(dp), intent (in), optional :: h
01169   real(dp), intent (in), optional :: z
01170   character(*), optional :: method
01171   real(dp) :: h_, z_, eps, t
01172   if (present(h)) then
01173     h_=h
01174   else
01175     h_=0.
01176   endif
01177   if (present(z)) then
01178     z_=z
01179   else
01180     z_=0.
01181   endif
01182   if (present(method) &
01183     .and. (method.eq."spotl" .or. method.eq."olsson")) then
01184     if(method.eq."spotl") then
01185       eps = h_/ earth%radius
01186       green_newtonian =                                    &
01187         1. /earth%radius**2                                &
01188         *(eps + 2. * (sin(psi/2.))**2 )                    &
01189         /((4.*(1.+eps)* (sin(psi/2.))**2 + eps**2)**(3./2.)) &
01190         * gravity%constant                                 &
01191         * green_normalization("f",psi=psi)
01192       return
01193     else if (method.eq."olsson") then
01194       t = earth%radius/(earth%radius +h_)
01195       green_newtonian =                   &
01196         1 / earth%radius**2 * t**2 *      &
01197         (1. - t * cos(psi) ) /           &
01198         ( (1-2*t*cos(psi) +t**2 )**(3./2.) ) &
01199         * gravity%constant                &
01200         * green_normalization("f",psi=psi)
01201       return
01202     endif
01203   else
01204     green_newtonian =                                                &
01205       ((earth%radius + h_) - (earth%radius + z_) * cos(psi))        &
01206       / ((earth%radius + h_)**2 + (earth%radius + z_)**2            &
01207       -2*(earth%radius + h_)*(earth%radius + z_)*cos(psi))**(3./2.)
01208
01209     green_newtonian = green_newtonian &
01210       * gravity%constant / earth%gravity%mean  * green_normalization("m", psi=psi)
01211     return
01212   endif
01213 end function
01214 end module
```

9.13 grat/src/mod_normalization.f90 File Reference

Data Types

- module mod_normalization

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(dble(1.))))
00021   else if (method.eq."m") then ! merriam normalization
00022     green_normalization =  &
00023         psi * 1e15 * earth%radius**2 * 2 * pi * (1.- cos(d2r(dble(1.))))
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/real_vs_standard.f90 File Reference

Functions/Subroutines

- program **real_vs_standard**

9.18 real_vs_standard.f90

```
00001 ! ============================================================================
00002 !> \file
00003 ! ============================================================================
00004 program real_vs_standard
00005 !  use mod_constants, only :dp
00006   use mod_parser
00007   use mod_data
00008 !  use mod_aggf,      only : geop2geom
00009   implicit none
00010
00011   real(dp) :: cpu(2)
00012
00013   call cpu_time(cpu(1))
00014
00015  call intro(program_calling="real_vs_standard")
00016
00017
00018 !    do ii = 1 , min(2,size(model))
00019 !      if (model(ii)%if) call get_variable ( model(ii) , date = dates(j)%date)
00020 !    enddo
00021 !
00022 !
00023 !
00024 !!\todo
00025 !    do i = 1 , size(sites)
00026 !      write(output%unit, '(2f15.5f)', advance ="no") sites(i)%lat ,sites(i)%lon
00027 !      iii=iii+1
00028 !!      call convolve (sites(i) , green , results(iii), denserdist = denser(1) , denseraz = denser(2))
00029 !      write (output%unit,'(15f13.5)') , results(iii)%e ,results(iii)%n  ,results(iii)%dt ,
       results(iii)%dh, results(iii)%dz
00030 !    enddo
00031 !  enddo
00032 !
00033 !
00034 !  call cpu_time(cpu_finish)
00035 !  write(log%unit, '(/,"Execution time:",1x,f16.9," seconds")') cpu_finish - cpu_start
00036 !  write(log%unit, form_separator)
00037 !
00038 !  print * , model(1)%level
00039 !  print *
00040 !  lat =00
00041 !  lon = 00
00042 !  call get_value(model(1),lat,lon, val(0))
00043 !
00044 !  do i =1, size(model(2)%level)
00045 !    call get_value(model(2),lat,lon, val(i), level = i, method=1)
00046 !  enddo
00047 !  print  '(2f10.2)', lat , lon ,  (val(i),geop2geom(val(i)/1000)*1000., i=0,size(model(2)%level))
00048 end program
```

9.19 grat/src/value_check.f90 File Reference

Functions/Subroutines

- program **value_check**

9.19.1 Detailed Description

Date

2013-01-09

Author

M. Rajner

Definition in file value_check.f90.

9.20 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)
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       write (output%unit , '(a10,1x,a14,1x)' , advance = "no" ) "#mjd", "date"
00045     endif
00046   endif
00047
00048   ! print header
00049   if (output%header.and.size(site).gt.0) then
00050     write (output%unit, '(a8,2a10$)') "name", "lat", "lon"
00051     if (output%height) then
00052       write (output%unit, '(a10$)') "height"
00053     endif
00054     if (output%level) then
00055       write (output%unit, '(a6$)') "level"
00056     endif
00057   endif
00058   do i = 1, size(model)
00059     if (output%header) then
00060       write (output%unit,'(a13)', advance='no') trim(model(i)%dataname)
00061     endif
00062   enddo
00063   if(output%header) write(output%unit, *)
00064
00065   do j = start, size(date)
00066     do i = 1 , size(model)
00067       if (model(i)%if) then
00068         if ( &
00069             .not.(model(i)%autoloadname.eq."ERA" &
00070             .and.(model(i)%dataname.eq."GP".or.model(i)%dataname.eq."VT")) &
00071             .and.(j.eq.1.and. model(i)%autoload &
00072             .or. (   &
00073             model(i)%autoload &
00074             .and. .not. date(j)%date(1).eq.date(j-1)%date(1)  &
00075             ) &
00076             ) &
00077             ) then
00078           call model_aliases(model(i), year=date(j)%date(1))
00079         else if ( &
00080             j.eq.1.and. model(i)%autoload &
00081             .or. (   &
00082             model(i)%autoload &
00083             .and. .not.( &
00084             date(j)%date(1).eq.date(j-1)%date(1)  &
```

```
00085           .and.date(j)%date(2).eq.date(j-1)%date(2) &
00086           ) &
00087           ) &
00088           ) then
00089         call model_aliases( &
00090             model(i), year=date(j)%date(1), month=date(j)%date(2))
00091       endif
00092       if (allocated(date).and.model(i)%exist) then
00093         call get_variable(model(i), date = date(j)%date)
00094       elseif (model(i)%exist) then
00095         call get_variable(model(i))
00096       endif
00097     endif
00098   enddo
00099
00100   ! print only dates if no site given
00101   if (j.gt.0 .and. size(site).lt.1) then
00102     if (dryrun) then
00103       write (output%unit , '(i4.4,5(i2.2),$)') date(j)%date
00104       if (j.lt.size(date)) write (output%unit , '(", ",$)')
00105     else
00106       write (output%unit , '(f10.3,1x,i4.4,5(i2.2))'  ) date(j)%mjd , date(j)%date
00107     endif
00108   endif
00109
00110   if (level%all.and..not.allocated(level%level)) then
00111     allocate(level%level(size(model(1)%level)))
00112     level%level=model(1)%level
00113   endif
00114
00115   if (size(level%level).lt.1) then
00116     start_level=0
00117   else
00118     start_level=1
00119   endif
00120
00121   do ilevel=start_level, size(level%level)
00122     do i = 1 , size(site)
00123       iprogress = iprogress + 1
00124       ! add time stamp if -D option was specified
00125       if (j.gt.0) then
00126         write (output%unit , '(f10.3,1x,i4.4,5(i2.2),1x)' , advance = "no" ) date(j)%mjd , date(j)%date
00127       endif
00128
00129       ! if this point should not be used (polygon) leave as zero
00130       if (allocated(polygon).and.polygon(1)%if) then
00131         call chkgon(site(i)%lon, site(i)%lat, polygon(1), iok)
00132       else
00133         iok=1
00134       endif
00135
00136       imodel = 0
00137       do ii = 1 , size (model)
00138         if (model(ii)%if.or.model(ii)%if_constant_value) then
00139           imodel = imodel + 1
00140           if (iok.eq.1) then
00141             if (j.eq.0) then
00142               call get_value(model(ii), site(i)%lat, site(i)%lon, val(imodel), &
00143                   method=info(1)%interpolation, level=level%level(ilevel))
00144             else
00145               call get_value(model(ii), site(i)%lat, site(i)%lon, val(imodel), &
00146                   method=info(1)%interpolation, date=date(j)%date, level=level%level(ilevel))
00147             endif
00148           else
00149             val(imodel) = 0
00150           endif
00151           if (model(ii)%dataname.eq."LS") val(ii)=int(val(ii))
00152         endif
00153       enddo
00154       write (output%unit , '(a8,2f10.4$)') site(i)%name, site(i)%lat, site(i)%lon
00155       if (output%height) then
00156         write (output%unit, '(f10.3$)') site(i)%height
00157       endif
00158
00159       if (output%level.and. allocated(level%level)) then
00160         write (output%unit, '(i6$)') level%level(ilevel)
00161       elseif(output%level) then
00162         write (output%unit, '(i6$)') ilevel
00163       endif
00164
00165       if (ind%model%tp.ne.0) then
00166         if (ind%model%gp.eq.0) call print_warning("need @GP with @TP")
00167         val(ind%model%tp)= &
00168             standard_pressure( &
00169             val(ind%model%gp), &
00170             use_standard_temperature=.true., &
00171             method = model(ind%model%tp)%name &
```

```
00172                )
00173
00174            if (output%rho) then
00175              val(ind%model%tp)=val(ind%model%tp)/(r_air * standard_temperature(val(ind%model%gp)))
00176            endif
00177
00178            val(ind%model%tp)= &
00179                variable_modifier(val(ind%model%tp),model(ind%model%tp)%datanames(1))
00180          endif
00181
00182          if (ind%model%tpf.ne.0) then
00183            if (any([ind%model%gp,ind%model%sp,ind%model%hp,ind%model%t].eq.0)) &
00184                call print_warning("not enough with @TPF")
00185            val(ind%model%tpf)= &
00186                standard_pressure( &
00187                val(ind%model%gp), &
00188                p_zero=val(ind%model%sp), &
00189                temperature=val(ind%model%t), &
00190                use_standard_temperature=.true., &
00191                h_zero=val(ind%model%hp), &
00192                method = model(ind%model%tpf)%name &
00193                )
00194            if (output%rho) then
00195              val(ind%model%tpf)=val(ind%model%tpf)/(r_air * standard_temperature(val(ind%model%gp)))
00196            endif
00197            val(ind%model%tpf)=variable_modifier(val(ind%model%tpf),model(ind%model%tpf)%datanames(1))
00198          endif
00199          if (ind%model%rho.ne.0) then
00200            if (any([ind%model%gp,ind%model%sp,ind%model%hp,ind%model%t,ind%model%vt].eq.0)) &
00201                call print_warning("not enough with @rho")
00202            val(ind%model%rho)= &
00203                100.*level%level(ilevel)/(r_air * val(ind%model%vt))
00204          endif
00205
00206          ! if (output%gp2h) then
00207            ! val(ind%model%gp) = &
00208                ! geop2geom( &
00209                ! val(ind%model%gp)  &
00210                ! / ( (1. -0.002637 *cos (2. * d2r(site(i)%lat))) &
00211                ! * earth%gravity%mean) &
00212                ! )
00213          ! endif
00214
00215          write (output%unit , "("// output%form // '$)') val
00216
00217          if (output%unit.ne.output_unit.and..not.quiet) then
00218            call cpu_time(cpu(2))
00219            call progress(100*iprogress/(max(size(date),1)*max(size(site),1)*max(size(level%level),1)), cpu(2
      )-cpu(1))
00220          endif
00221          if (size(val).gt.0) write (output%unit , *)
00222        enddo
00223      enddo
00224    enddo
00225
00226    if (ind%moreverbose%d.ne.0) then
00227      do i=1, size(model)
00228        do j=1, size(model(i)%time)
00229          write (moreverbose(ind%moreverbose%d)%unit, '(g0,1x,i4,5i2.2)') &
00230              model(i)%time(j), model(i)%date(j,:)
00231        enddo
00232      enddo
00233    endif
00234
00235    if (ind%moreverbose%j.ne.0) then
00236      do i = 1, size(model)
00237        do j = 1, size(model(i)%level)
00238          write (moreverbose(ind%moreverbose%j)%unit, '(i5)') &
00239              model(i)%level(j)
00240        enddo
00241      enddo
00242    endif
00243
00244    call cpu_time(cpu(2))
00245    if (output%unit.ne.output_unit.and..not.quiet) then
00246      call progress(100*iprogress/(max(size(date),1)*max(size(site),1)*max(size(level%level),1)), cpu(2)-cpu(
      1), every=1)
00247      close(output_unit)
00248    endif
00249    write(log%unit, '(/,"Execution time:",1x,f16.9," seconds")') cpu(2)-cpu(1)
00250    write(log%unit, form_separator)
00251
00252 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/tmp")
00024
00025   call compute_tabulated_green_functions('/home/mrajner/src/grat/dat/rajner_green_full.dat'  , method=
      "full"        , predefined=.false.)
00026   call compute_tabulated_green_functions('/home/mrajner/src/grat/dat/rajner_green_rough.dat' , predefined=.
      false., rough=.true.)
00027   call compute_tabulated_green_functions('/home/mrajner/src/grat/dat/rajner_green_simple.dat', method=
      "simple"      , predefined=.false.)
00028   call compute_tabulated_green_functions('/home/mrajner/src/grat/dat/rajner_green.dat'       , predefined=.
      false. )
00029   call aggf_resp_fels_profiles('/home/mrajner/src/grat/examples/aggf_resp_fels_profiles.dat')
00030   call mass_vs_height('/home/mrajner/src/grat/examples/mass_vs_height.dat')
00031   call aggf_resp_hmax('/home/mrajner/src/grat/examples/aggf_resp_zmax.dat')
00032   call aggf_resp_dz('/home/mrajner/src/grat/examples/aggf_resp_dz.dat')
00033   call aggf_resp_t('/home/mrajner/src/grat/examples/aggf_resp_t.dat')
00034   call aggf_resp_h('/home/mrajner/src/grat/examples/aggf_resp_h.dat')
00035
00036   call cpu_time(cpu(2))
00037   print '("Total time: ",f8.3,x,"[s]")', cpu(2)-cpu(1)
00038
00039 contains
00040 ! =============================================================================
00041 !> Mass of atmosphere respect to height
00042 ! =============================================================================
00043 subroutine mass_vs_height (filename)
00044   use, intrinsic:: iso_fortran_env
00045   use mod_utilities, only: file_exists
00046   use mod_constants, only : dp, pi, earth, r_air
00047   use mod_atmosphere
00048   character(*) , intent (in) , optional:: filename
00049   real(dp) :: max_height,dh, percent
00050   real(dp) , allocatable, dimension(:):: mass, height
00051   integer::i,j,file_unit
00052
00053   if (present(filename)) then
00054     if (file_exists(filename)) return
00055     open ( &
```

```
00056     newunit = file_unit, &
00057     file    = filename,  &
00058     action  = 'write'   &
00059     )
00060   else
00061     file_unit = output_unit
00062   endif
00063   write(*,*), "mass_vs_height ---> ",filename
00064
00065   max_height=50000.
00066   dh=10
00067
00068   allocate(height(int(max_height/dh)+1))
00069   allocate(mass(size(height)))
00070   do i =1,size(height)
00071     height(i) = dh*(i-1)
00072     mass(i) = standard_pressure( &
00073         height(i), &
00074         method="standard", &
00075         use_standard_temperature=.true., &
00076         nan_as_zero=.true.) &
00077         / (r_air * standard_temperature(height(i)))
00078   enddo
00079
00080   do i =0,50000,1000
00081    percent=0
00082    do j = 1 , size(height)
00083      if (height(j).le.dble(i)) percent=percent+mass(j)
00084    enddo
00085    percent = percent / sum(mass)  * 100.
00086    write(file_unit, '(i6,2f19.9,es10.3)' ) , i ,percent , &
00087    100-(earth%radius+dble(1))**2 &
00088    * standard_pressure(dble(i),method="standard", use_standard_temperature=.true.) &
00089    / standard_gravity(dble(i))&
00090    /earth%radius**2/standard_pressure(dble(0),method="standard") * standard_gravity(dble(0))*100
00091   enddo
00092 end subroutine
00093
00094 ! ============================================================================
00095 !> Reproduces data to Fig.~3 in \cite Warburton77
00096 !!
00097 !! \date 2013-03-18
00098 !! \author M. Rajner
00099 !!
00100 ! ============================================================================
00101 subroutine simple_atmospheric_model (filename)
00102   use, intrinsic:: iso_fortran_env
00103   use mod_utilities, only: file_exists
00104   use mod_constants
00105   use mod_aggf, only:simple_def, bouger
00106
00107   real(dp) :: r ! km
00108   integer :: file_unit
00109   character(*) , intent (in) , optional:: filename
00110   real(dp) :: h =9.
00111
00112   if (present(filename)) then
00113     if (file_exists(filename)) return
00114     open ( &
00115     newunit = file_unit, &
00116     file    = filename,  &
00117     action  = 'write'    &
00118     )
00119   else
00120     file_unit = output_unit
00121   endif
00122
00123   write(*,*), "simple_atmospheric_model ---> ",filename
00124
00125   do r = 0. , 25*8
00126     write (file_unit,  * ), r,-100*bouger(h=h,r=r)/(earth%gravity%mean*h)  * 1e8, & !conversion to
   microGal
00127     -simple_def(r) * 1e8
00128   enddo
00129
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 (filename, method, dz, &
00139   predefined,fels_type, rough)
00140   use mod_constants, only: dp
00141   use mod_aggf ,     only: aggf, aggfd
```

```
00142   use mod_green,     only: green
00143   use mod_utilities, only: d2r, file_exists
00144   use mod_atmosphere
00145
00146   integer :: i, file_unit
00147   character(*), intent(in) :: filename
00148   real(dp), optional :: dz
00149   character(*), optional :: fels_type
00150   character(*), optional :: method
00151   logical, optional, intent(in) :: predefined, rough
00152
00153   if (file_exists(filename)) then
00154     return
00155   else
00156     print '(a,a)', "compute_tabulated_green_functions --> ", trim(filename)
00157   endif
00158
00159   call get_green_distances
00160
00161   open (                  &
00162   newunit = file_unit, &
00163   file    = filename,  &
00164   action  = 'write'    &
00165   )
00166
00167   !print header
00168   write ( file_unit,*) '# This is set of AGGF computed using module ', &
00169   'aggf from grat software'
00170   write ( file_unit,*) '# Normalization according to Merriam92'
00171   write ( file_unit,*) '# Marcin Rajner'
00172   write ( file_unit,*) '# For detail see www.geo.republika.pl'
00173   write ( file_unit,'(10(a23))')  '#psi[deg]',                            &
00174   'GN[microGal/hPa]'     , 'GN/dT[microGal/hPa/K]' ,                   &
00175   'GN/dh[microGal/hPa/m]', 'GN/dz[microGal/hPa/m]'
00176
00177   do i= 1, size(green(1)%distance)
00178     write(file_unit, '(13f15.6)'), &
00179     green(1)%distance(i), &
00180     aggf(d2r(green(1)%distance(i)), method=method, dz=dz                           , predefined=
      predefined, fels_type=fels_type, rough=rough), &
00181     aggfd(d2r(green(1)%distance(i)), method=method, dz=dz, aggfdt=.true.            , predefined=
      predefined, fels_type=fels_type, rough=rough), &
00182     aggf(d2r(green(1)%distance(i)), method=method, dz=dz, first_derivative_h=.true., predefined=
      predefined, fels_type=fels_type, rough=rough), &
00183     aggf(d2r(green(1)%distance(i)), method=method, dz=dz, first_derivative_z=.true., predefined=
      predefined, fels_type=fels_type, rough=rough)
00184   enddo
00185
00186   close(file_unit)
00187 end subroutine
00188
00189 ! ============================================================================
00190 !> Compare different vertical temperature profiles impact on AGGF
00191 ! ============================================================================
00192 subroutine aggf_resp_fels_profiles (filename)
00193   use mod_constants, only: dp
00194   use mod_aggf, only : aggf
00195   use mod_green, only: green
00196   character (len=255) ,dimension (6) :: fels_types
00197   integer :: i , j, file_unit
00198   character(*), intent(in), optional :: filename
00199
00200   if (present(filename)) then
00201     if (file_exists(filename)) return
00202     open ( newunit = file_unit , &
00203       file =filename , &
00204       action  = 'write' )
00205   else
00206     file_unit = output_unit
00207   endif
00208     print *, "aggf_resp_fels_profiles -->", filename
00209
00210   ! Get the spherical distances from Merriam92
00211   call get_green_distances()
00212
00213   ! ! All possible optional arguments for standard_temperature
00214   fels_types = (/ &
00215     "US1976"             , "tropical",    &
00216     "subtropical_summer" , "subtropical_winter" , &
00217     "subarctic_summer"   , "subarctic_winter"     &
00218     /)
00219   ! print header
00220   write (file_unit, '(100(a20))') &
00221     'psi', (trim(fels_types(i)), i = 1, size(fels_types))
00222
00223   ! print results
00224   do i = 1, size(green(1)%distance)
```

```
00225     write(file_unit, '(<size(fels_types)+1>f20.5)'), &
00226       green(1)%distance(i), &
00227       (aggf( &
00228       d2r(green(1)%distance(i)), &
00229       method="standard", &
00230       fels_type=fels_types(j)), j=1,size(fels_types) &
00231       )
00232   enddo
00233   close(file_unit)
00234 end subroutine
00235
00236
00237 !! ============================================================================
00238 !!> Compare different vertical temperature profiles
00239 !!!
00240 !!! Using tables and formula from \cite Fels86
00241 !!! \author M. Rajner
00242 !!! \date 2013-03-19
00243 !! ============================================================================
00244 subroutine compare_fels_profiles (filename)
00245   use iso_fortran_env
00246   use mod_utilities, only: file_exists
00247   use mod_constants, only: dp
00248   use mod_atmosphere, only : standard_temperature
00249   character (len=255) ,dimension (6) :: fels_types
00250   real (dp) :: height
00251   integer :: i , file_unit , i_height
00252   character(*), intent (in),optional:: filename
00253
00254   ! All possible optional arguments for standard_temperature
00255   fels_types = (/ "US1976"             , "tropical",    &
00256     "subtropical_summer" , "subtropical_winter" , &
00257     "subarctic_summer"   , "subarctic_winter"    /)
00258
00259   if (present(filename)) then
00260     if (file_exists(filename)) return
00261     open ( newunit = file_unit , &
00262       file =filename , &
00263       action  = 'write' )
00264   else
00265     file_unit = output_unit
00266   endif
00267
00268   print * , "compare_fels_profiles --->", filename
00269
00270   ! Print header
00271   write ( file_unit , '(100(a20))' ) &
00272     'height', ( trim( fels_types(i) ) , i = 1 , size (fels_types) )
00273
00274   ! Print results
00275   do i_height = 0 , 70 , 1
00276     height=dble(i_height)
00277     write ( file_unit , '(f20.3$)' ) , height
00278     do i = 1 , size (fels_types)
00279       write ( file_unit , '(f20.3$)' ),  standard_temperature(height*1000, fels_type=fels_types(i))
00280     enddo
00281     write ( file_unit , * )
00282   enddo
00283   close(file_unit)
00284 end subroutine
00285
00286 ! ============================================================================
00287 !> Computes AGGF for different site height (h)
00288 ! ============================================================================
00289 subroutine aggf_resp_h (filename)
00290 !  use mod_constants, only : dp
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
00307  call get_green_distances()
00308
00309  heights=[0.,1.,10.,100.,1000.,10000.]
00310
00311
```

```
00312 write (file_unit, "(a12,6(x,'h',f0.0))") "distance", heights(1:6)
00313  do i =1 , size (green(1)%distance)
00314    ! denser sampling
00315    do ii = 0,8
00316      aux  = green(1)%distance(i) + ii * (green(1)%distance(i+1) - green(1)%distance(i)) / 9.
00317      if (aux.gt.0.2 ) exit
00318      write (file_unit, '(F12.6$)') , aux
00319      do j =  1 , size(heights)
00320        write (file_unit,'(f12.4,1x,$)') aggf(d2r(aux), method="standard", h=heights(j))
00321      enddo
00322      write (file_unit,*)
00323    enddo
00324  enddo
00325  close (file_unit)
00326 end subroutine
00327
00328 ! ============================================================================
00329 !> This computes AGGF for different surface temperature
00330 !!
00331 !! \author M. Rajner
00332 !! \date 2013-03-18
00333 ! ============================================================================
00334 subroutine aggf_resp_t (filename)
00335   use mod_green, only: green
00336   ! use mod_constants, only : dp , atmosphere
00337   use mod_aggf, only : aggf
00338   real(dp), dimension(:,:), allocatable :: results
00339   integer :: i, j
00340   character(*), intent(in), optional :: filename
00341   integer :: file_unit
00342   real(dp) :: temperatures(3)
00343
00344   if (present(filename)) then
00345     if (file_exists(filename)) return
00346     open ( newunit = file_unit , &
00347         file =filename , &
00348         action  = 'write' )
00349   else
00350     file_unit = output_unit
00351   endif
00352   call get_green_distances()
00353
00354   allocate(results(size(green(1)%distance), 3))
00355
00356   temperatures=[0., 15., -45]
00357
00358   write(file_unit, '(4a12)') "distance","T0+0", "T0+15", "T0-45"
00359   do i = 1 , size(green(1)%distance)
00360     write(file_unit, '(f12.5$)') green(1)%distance(i)
00361     do j=1, size(temperatures)
00362       write(file_unit, '(f12.5$)') &
00363         aggf(d2r(green(1)%distance(i)), method="standard", t_zero=temperatures(j))
00364     enddo
00365     write(file_unit, *)
00366   enddo
00367   close (file_unit)
00368 end subroutine
00369
00370 ! ============================================================================
00371 !> \brief This computes AGGF for different height integration step
00372 ! ============================================================================
00373 subroutine aggf_resp_dz (filename)
00374   use mod_green
00375   use mod_aggf, only: aggf
00376   real(dp), dimension(:,:), allocatable :: results
00377   real(dp), dimension(:), allocatable :: dzs
00378
00379   integer :: file_unit, i, j
00380   character(*) , intent (in) , optional:: filename
00381
00382   if (present(filename)) then
00383     if (file_exists(filename)) return
00384     open ( newunit = file_unit , &
00385         file =filename , &
00386         action  = 'write' )
00387   else
00388     file_unit = output_unit
00389   endif
00390
00391   call get_green_distances()
00392
00393   allocate(dzs(5))
00394   dzs=(/ 0.01, 0.1, 1., 10., 100./)
00395
00396   allocate (results(size(green(1)%distance(1:29)),size(dzs)))
00397   results = 0.
00398
```

```
00399   do i = 1 , size (results(:,1))
00400     do j=1,size(dzs)
00401       results(i,j)=i+j
00402       results(i,j)=aggf(d2r(green(1)%distance(i)), &
00403           method="standard", &
00404           dz=dzs(j))
00405     enddo
00406     ! compute relative errors from column 2 for all dz with respect to column 1
00407     results(i,2:) = abs((results(i,2:) - results(i,1)) / results(i,1)*100.  )
00408   enddo
00409
00410   write(file_unit, '(a14,<size(dzs)>f14.4)') "psi_dz", dzs
00411   write(file_unit, '(f14.5,<size(dzs)>e14.4)') &
00412       (green(1)%distance(i), results(i,:), i=1,size(results(:,1)))
00413   close(file_unit)
00414 end subroutine
00415
00416 ! ============================================================================
00417 !> \brief This computes standard atmosphere parameters
00418 !!
00419 !! It computes temperature, gravity, pressure, pressure (simplified formula)
00420 !! density for given height
00421 ! ============================================================================
00422 subroutine standard1976(filename)
00423   use, intrinsic :: iso_fortran_env
00424   use mod_utilities, only: file_exists
00425   use mod_constants, only : dp, r_air
00426   use mod_atmosphere, only: &
00427       standard_temperature, standard_pressure , &
00428       standard_gravity,      standard_density
00429   integer :: file_unit
00430   character(*) , intent (in) , optional:: filename
00431   real(dp) :: height
00432
00433   if (present(filename)) then
00434     if (file_exists(filename)) return
00435     open ( newunit = file_unit , &
00436         file =filename , &
00437         action  = 'write' )
00438   else
00439     file_unit = output_unit
00440   endif
00441
00442   print * , "standard atmosphere --->", filename
00443   ! print header
00444   write ( file_unit , '(6(a15))' ) &
00445       'height', 'T' , 'g' , 'p', 'rho'
00446   do height=0.,68000. , 1000
00447     ! print results to file
00448     write( file_unit,'(5f15.5, e12.3)'), &
00449         height/1000.,                         &
00450         standard_temperature(height),         &
00451         standard_gravity(height),             &
00452         standard_pressure(height, method="standard")/100.,       &  ! --> hPa
00453         standard_pressure(height, method="standard") &
00454         /(r_air*standard_temperature(height))
00455   enddo
00456   close( file_unit )
00457 end subroutine
00458
00459 ! ============================================================================
00460 !> \brief This computes relative values of AGGF for different atmosphere
00461 !! height integration
00462 ! ============================================================================
00463 subroutine aggf_resp_hmax (filename)
00464   use mod_utilities, only: file_exists, logspace, d2r
00465   ! use mod_constants, only : dp
00466   use mod_aggf, only : aggf
00467   real (dp) , dimension (2) :: psi
00468   real (dp) , dimension (:)    , allocatable :: heights
00469   real (dp) , dimension (:,:) , allocatable :: results
00470   integer :: file_unit, n, i, j
00471   character(*) , intent (in) , optional:: filename
00472
00473   if (present(filename)) then
00474     if (file_exists(filename)) return
00475     open ( newunit = file_unit , &
00476         file =filename , &
00477         action  = 'write' )
00478   else
00479     file_unit = output_unit
00480   endif
00481
00482   print * , "standard atmosphere ---> ", filename
00483   psi=(/0.0001, 10 /)
00484
00485   n=90
```

```
00486   allocate(heights(n))
00487
00488   heights= logspace(real(1e-1,dp), real(60000,dp),n)
00489
00490   allocate (results(size(heights), size(psi)))
00491   results=0
00492
00493   do j=1, size(heights)
00494     do i = 1, size(psi)
00495       results(j,i) =aggf(d2r(psi(i)),method="standard", zmax=heights(j))
00496     enddo
00497   enddo
00498   do i = 1, size(psi)
00499     results(:,i)=results(:,i)/results(size(heights),i) * 100. ! in %
00500   enddo
00501
00502   write(file_unit , '(a14,SP,100f14.5)' ),"#heght\psi", (psi(j) , j= 1,size(psi))
00503   do i=1, size (results(:,1))
00504     write(file_unit, '(100f14.4)' ) heights(i)/1000, (results(i,j), j = 1, size(psi) )
00505   enddo
00506   close(file_unit)
00507 end subroutine
00508
00509 ! ============================================================================
00510 ! ============================================================================
00511 subroutine aggf_thin_layer (filename)
00512   use, intrinsic:: iso_fortran_env
00513   use mod_constants, only : dp , pi
00514   use mod_aggf, only : gn_thin_layer
00515   use mod_utilities, only: d2r, file_exists
00516   use mod_green
00517
00518   integer :: file_unit , i
00519   character(*) , intent (in) , optional:: filename
00520
00521   if (file_exists(filename)) return
00522
00523   call get_green_distances()
00524
00525   write(*,*), "aggf_thin_layer ---> ",filename
00526   if (present(filename)) then
00527     open (newunit = file_unit , &
00528         file =filename , &
00529         action  = 'write' )
00530   else
00531     file_unit = output_unit
00532   endif
00533   do i = 1 , size (green(1)%distance)
00534     write(file_unit,*) green(1)%distance(i) ,green(1)%data(i), &
00535         gn_thin_layer(d2r(green(1)%distance(i)))
00536   enddo
00537 end subroutine
00538
00539 ! =============================================================================
00540 ! =============================================================================
00541 subroutine admit_niebauer(filename)
00542   use mod_constants
00543   use mod_utilities
00544   real(dp) :: a
00545   real(dp) :: theta
00546   real(dp) :: b , f
00547   character(*), intent(in) :: filename
00548   integer::iun
00549
00550   if (file_exists(filename)) return
00551   print * , "admit_niebauer ---> ", filename
00552
00553   open (newunit=iun, file=filename, action = 'write')
00554
00555   f=earth%radius/9500
00556   do theta=0.5 , 180, 0.01
00557     b= 2*f*sin(d2r(theta/2))
00558     a= 2*pi * gravity%constant / earth%gravity%mean* &
00559         (1 - b/(2*f) -1/b + 2/f)
00560     write(iun, *) , theta , a *1e10
00561   enddo
00562 end subroutine
00563
00564 ! =============================================================================
00565 !> compute green newtonian function
00566 ! =============================================================================
00567 subroutine green_newtonian_compute(filenames)
00568   use mod_utilities, only: file_exists
00569   use mod_green
00570   use mod_utilities, only : logspace , d2r
00571   integer:: iun , n , i , j , k
00572   real (dp) , allocatable , dimension(:) :: psi , h
```

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