

Rheolef, the finite element system.

Reference manual
Version 7.0
18 February 2018

by Pierre Saramito

Copyright © 1996-2018 by Pierre Saramito.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

rheolef is a computer environment that serves as a convenient *laboratory* for computations involving finite element-like methods. It provides a set of unix commands and C++ algorithms and containers. This environment is currently under development.

Algorithms are related, from one hand to *sparse matrix* basic linear algebra, e.g. $x+y$, $A*x$, $A+B$, $A*B$, ... From other hand, we focus development on preconditionned linear and non-linear solver e.g. direct and iterative methods for $A*x=b$, iterative solvers for Stokes and Bingham flows problems. Future developments will consider also viscoelastic flows problems.

Containers covers first the classic graph data structure for sparse matrix formats and finite element *meshes*. An higher level of abstraction is provided by containers related to approximate finite element *spaces*, discrete *fields* and bilinear *forms*. Current developments concerns distributed sparse matrix. Future developments will consider also 3D anisotropic *adaptative* mesh generation.

Please send all comments and bug reports by electronic mail to `Pierre.Saramito@imag.fr`.

Table of Contents

1	Installing rheolef.....	3
1.1	Reading the documentation	3
1.2	Installing from source as root user	3
1.3	Using and alternative installation directory	4
1.4	Required libraries	4
1.5	Optional highly recommended libraries	4
1.6	Optional distributed and parallel libraries	5
1.7	Run-time optional tools	5
1.8	Build-time extra tools for rheolef developers	5
1.9	Portability issues	6
1.10	Configure command line options	6
1.11	Configure command line options	7
1.12	The directory structure	9
2	Reporting bugs	11
3	Commands	13
3.1	<code>branch</code> – handle a family of fields	13
3.2	<code>field</code> – plot a field	15
3.3	<code>geo</code> – plot a finite element mesh	20
3.4	<code>basis</code> – view a basis	26
3.5	<code>bamg2geo</code> – convert bamg mesh in geo format	28
3.6	<code>mkgeo_ball</code> – build an unstructured mesh of an ellipsoid, in 2d or 3d	29
3.7	<code>mkgeo_contraction</code> – build an unstructured 2d mesh of an abrupt contraction	31
3.8	<code>mkgeo_grid</code> – build a structured mesh of a parallelotope, in 1d, 2d or 3d	32
3.9	<code>mkgeo_sector</code> – build an unstructured 2d mesh of a sector	35
3.10	<code>mkgeo_ugrid</code> – build an unstructured mesh of a parallelotope, in 1d, 2d or 3d	36
3.11	<code>msh2geo</code> – msh2geo – convert gmsh mesh in geo format ...	38
3.12	<code>bamg</code> – bidimensional anisotropic mesh generator	39
3.13	<code>rheolef-config</code> – get installation directories	47

4	Classes	51
4.1	<code>band</code> - compute the band around a level set	51
4.2	<code>branch</code> - a parameter-dependent sequence of field	52
4.3	<code>characteristic</code> - the Lagrange-Galerkin method implemented	54
4.4	<code>continuation_option</code> - send options to the continuation algorithm	55
4.5	<code>field</code> - piecewise polynomial finite element field	56
4.6	<code>field_functor</code> - a functor wrapper suitable for field expressions	63
4.7	<code>form</code> - representation of a finite element bilinear form	64
4.8	<code>functor</code> - a function wrapper suitable for field expressions	68
4.9	<code>geo</code> - finite element mesh	69
4.10	<code>integrate_option</code> - send options to the integrate function	78
4.11	<code>space</code> - piecewise polynomial finite element space	79
4.12	<code>test, trial</code> - symbolic arguments in variational expressions	83
4.13	<code>ad3</code> - automatic differentiation with respect to variables in \mathbb{R}^3	83
4.14	<code>basis</code> - polynomial basis	84
4.15	<code>basis_option</code> - options of the finite element space	87
4.16	<code>point</code> - vertex of a mesh	89
4.17	<code>quadrature_option</code> - send options to the integrate function	94
4.18	<code>tensor</code> - a $N \times N$ tensor, $N=1,2,3$	95
4.19	<code>tensor3</code> - a third order tensor	98
4.20	<code>tensor4</code> - a fourth order tensor	100
4.21	<code>asr</code> - associative sparse matrix	101
4.22	<code>csr</code> - compressed sparse row matrix	103
4.23	<code>dia</code> - diagonal matrix	110
4.24	<code>disarray</code> - container in distributed environment	111
4.25	<code>distributor</code> - data distribution table	117
4.26	<code>idiststream, odiststream</code> - distributed interface for large data streams	119
4.27	<code>environment</code> - rheolef initialization	122
4.28	<code>eye</code> - the identity matrix	122
4.29	<code>solver</code> - direct or iterative solver interface	123
4.30	<code>solver_abtb</code> - direct or iterative solver interface for mixed linear systems	125
4.31	<code>solver_option</code> - options for direct or iterative solvers	127
4.32	<code>vec</code> - vector in distributed environment	130
4.33	<code>irheostream, orheostream</code> - large data streams	132

5	Algorithms	137
5.1	<code>adapt</code> - mesh adaptation.....	137
5.2	<code>compose</code> - a n-ary function with n fields.....	139
5.3	<code>continuation</code> - continuation algorithm for nonlinear problems.....	139
5.4	<code>damped_newton</code> - damped Newton nonlinear algorithm...	141
5.5	<code>integrate</code> - integrate a function or an expression.....	141
5.6	<code>interpolate</code> - Lagrange interpolation of a function.....	143
5.7	<code>level_set</code> - compute a level set from a function.....	143
5.8	<code>limiter</code> - discontinuous Galerkin slope limiter	144
5.9	<code>newton</code> - Newton nonlinear algorithm	145
5.10	<code>riesz</code> - approximate a Riesz representer	146
5.11	<code>cg</code> - conjugate gradient algorithm.	148
5.12	<code>diag</code> - get diagonal part of a matrix	150
5.13	<code>gmres</code> - generalized minimum residual method	150
5.14	<code>minres</code> - conjugate gradient algorithm.	154
5.15	<code>cg_abtb</code> , <code>cg_abtbc</code> , <code>minres_abtb</code> , <code>minres_abtbc</code> - solvers for mixed linear problems	156
5.16	<code>uzawa</code> - Uzawa algorithm.	157
5.17	<code>catchmark</code> - iostream manipulator.....	159
6	Internal classes	161
6.1	<code>domain_indirect</code> - a named part of a finite element mesh	161
6.2	<code>geo_domain</code> - a named part of a finite element mesh that behaves as a mesh	163
6.3	<code>geo_domain_indirect_rep</code> - a named part of a finite element mesh	164
6.4	<code>numbering</code> - global degree of freedom numbering	164
6.5	<code>basis_on_pointset</code> - pre-evaluated polynomial basis	166
6.6	<code>Bk</code> - Bernstein polynomial basis	167
6.7	<code>Pk</code> - Lagrange polynomial basis.....	167
6.8	<code>RTk</code> - The Raviart-Thomas vector-valued polynomial basis	168
6.9	<code>Sk</code> - Dubiner-Sherwin-Karniadakis polynomial basis.....	168
6.10	<code>edge</code> - Edge reference element	169
6.11	<code>geo_element</code> - element of a mesh.....	169
6.12	<code>hack_array</code> - container in distributed environment.....	170
6.13	<code>hexahedron</code> - Hexaedron reference element	175
6.14	<code>point</code> - Point reference element	177
6.15	<code>prism</code> - Prism reference element	177
6.16	<code>quadrangle</code> - Quadrangular reference element	178
6.17	<code>quadrature</code> - quadrature formulae on the reference lement	180
6.18	<code>reference_element</code> - reference element.....	181
6.19	<code>tetrahedron</code> - Tetraedron reference element	183
6.20	<code>triangle</code> - Triangle reference element	185

6.21	<code>dis_cpu_time</code> , <code>dis_wall_time</code> , <code>seq_cpu_time</code> , <code>seq_walltime</code> – Time in seconds since an arbitrary time in the past.....	186
6.22	<code>dis_memory_usage</code> , <code>seq_memory_usage</code> – physical memory in use.....	187
6.23	<code>index_set</code> - a set of indexes.....	188
6.24	<code>pair_set</code> - a set of (index,value) pair.....	189
6.25	<code>Vector</code> - STL <code>vector<T></code> with reference counting.....	190
6.26	<code>heap_allocator</code> - heap-based allocator.....	192
6.27	<code>smart_pointer</code> , <code>smart_pointer_clone</code> - reference counted safe pointer with true copy semantic.....	194
6.28	<code>stack_allocator</code> - stack-based allocator.....	197
7	Internal algorithms	201
7.1	<code>iorheo</code> - input and output functions and manipulation...	201
7.2	<code>typename_macro</code> , <code>pretty_typename_macro</code> - type demangler and pretty printer	204
8	Internal others	207
8.1	<code>acinclude</code> – autoconf macros.....	207
9	FAQ for developers	215
9.1	How to regenerate the <code>configure</code> script	215
9.1.1	In which order does the things build ?.....	215
9.1.2	What means these commands ?.....	215
9.2	How to save my version ?.....	216
9.2.1	Easy: no conflicts with another developer.....	216
9.2.2	I have conflicts with another developer	216
9.2.3	I have deleted a source file.....	217
Appendix @ The GNU General Public License		219
Concept Index.....		231
Program Index		233
Class Index.....		235
Approximation Index		237
Function Index		239
File Format Index.....		241
Related Tool Index		243

1 Installing rheolef

(Source file: ‘configure.ac’)

1.1 Reading the documentation

Rheolef comes with three documentations:

- an users guide with a set of complete examples (two volumes)
- a reference manual for unix commands an C++ classes
- the full browsable source code in html format

All these documentations are downloadable in various formats from the Rheolef home page. These files are also locally installed during the `make install` stage. See

```
rheolef-config --docdir
```

The users guide is generated in pdf format:

```
evince http://ljk.imag.fr/membres/Pierre.Saramito/rheolef/rheolef.pdf
evince http://ljk.imag.fr/membres/Pierre.Saramito/rheolef/rheolef2-dg.pdf
```

The reference manual is generated in html format:

```
firefox http://ljk.imag.fr/membres/Pierre.Saramito/rheolef/rheolef-refman.html
```

and you could add it to your bookmarks. Moreover, after performing `make install`, unix manuals are available for commands and classes:

```
man field
man 2 field
```

The browsable source code is generated in html format:

```
firefox http://ljk.imag.fr/membres/Pierre.Saramito/rheolef/source_html/files.html
```

1.2 Installing from source as root user

```
./configure
make
sudo make install
```

By default, the installation directory is ‘/usr/local’. Commands are in ‘/usr/local/bin’, libraries in ‘/usr/local/lib’, header files in ‘/usr/local/include’ and others file in ‘/usr/local/share’.

1.3 Using and alternative installation directory

The default install prefix is `‘/usr/local’`. If you are not the root user, or do not have its password, you could prefer an alternative directory, e.g. `‘HOME/sys’`. You should enter:

```
./configure --prefix=$HOME/sys
```

In that case, you have to add the following lines at the end of your `‘.profile’` or `‘.bash_profile’` file:

```
export PATH="$HOME/sys/bin:$PATH"
export LD_LIBRARY_PATH="$HOME/sys/lib:$LD_LIBRARY_PATH"
export MANPATH="$HOME/sys/man:$MANPATH"
```

Assuming you are using the `sh` or the `bash` unix interpreter. Conversely, if you are using `csh` or `tcsh`, add at the bottom of your `‘.cshrc’` file:

```
set path = ($HOME/sys/bin $path)
setenv LD_LIBRARY_PATH "$HOME/sys/lib:$LD_LIBRARY_PATH"
setenv MANPATH "$HOME/sys/man:$MANPATH"
```

If you are unsure about unix interpreter, get the `SHELL` value:

```
echo $SHELL
```

Then, you have to `source` the modified file, e.g.:

```
source ~/.bashrc
```

hpux users should replace `LD_LIBRARY_PATH` by `SHLIB_PATH`. If you are unsure, get the shared library path variable:

```
rheolef-config --shlibpath-var
```

Check also your installation by compiling a sample program with the installed library:

```
make installcheck
```

Since it is a common mistake, you can later, at each time, test your run-time environment sanity with:

```
rheolef-config --check
```

1.4 Required libraries

The following required libraries are available as precompiled packages under Debian GNU/Linux and Ubuntu systems.

- a) The `boost` library. It provides useful extensions of the `c++` standard template library. See <http://www.boost.org>.
- b) The `armadillo` library. It provides efficient `c++` linear algebra subroutines. See <http://arma.sourceforge.net>.

1.5 Optional highly recommended libraries

- a) The `umfpack` library provides a multifrontal sparse direct solver. Rheolef solvers run faster when using this library. See <http://faculty.cse.tamu.edu/davis/suitesparse.h>.

- b) The `cgal` library provide an easy access to efficient and reliable geometric algorithms. With it, Rheolef implements the characteristic method and interpolation from one mesh to another. This library is optional, but allows, when available, extended Rheolef features. See <http://www.cgal.org>.

1.6 Optional distributed and parallel libraries

With the optional following libraries, Rheolef is able to run in parallel and with distributed memory. By default, the Rheolef `configure` script tries to autodetect these libraries and be able to run in parallel. When these libraries are not found, the distributed features are turned off. If your libraries are installed in a non-standard location, you should indicate it by a suitable `configure` option. See below for these options.

- a) Rheolef bases on `mpi`, the message passing interface standard.
- b) `mumps` is a massively distributed sparse direct solver. The use of implicit numerical schemes leads to the resolution of large sparse linear systems. Rheolef installation optionally requires `mumps` is available at <http://graal.ens-lyon.fr/MUMPS>.
- c) Rheolef bases either on `scotch` or `metis` for distributed mesh partitioning. `scotch` is available at <http://www.labri.fr/perso/pelegrin/scotch>. `metis` is available at <http://www.cs.umn.edu/~metis>.
- d) Rheolef installation optionally uses `trilinos/ifpack` as a library of preconditioner used for iterative solvers. `trilinos` is available at <http://trilinos.sandia.gov>. When `trilinos` is not available, a less efficient builtin IC0 preconditioner is used.

1.7 Run-time optional tools

Using Rheolef suggests the use of the following optional tools for pre- and post-processing purpose:

```
gmsb
gnuplot
paraview
mayavi2
```

None of these tools are required when building and installing Rheolef.

1.8 Build-time extra tools for rheolef developers

These tools are required if you build from the development version (under CVS). Regenerating some source files, as initiated with the "bootstrap" command, requires auto-tools `autoconf`, `automake` and `libtool` and also `flex` and `bison`. It requires also `ginac` for polynomial basis management. `ginac` is available at <http://www.ginac.de>. It requires finally latex figure management tools to. Here is the `apt-get` commands to install these tools under Debian GNU/Linux and Ubuntu:

```
apt-get install cvs autoconf automake libtool
apt-get install flex bison
apt-get install pkg-config libginac-dev ginac-tools
apt-get install ghostscript gnuplot xfig transfig imagemagick graphviz
```

1.9 Portability issues

Rheolef is based on STL and its extensions, the BOOST library. Therefore you need a compiler that supports the ANSI C++ standards. STL and Rheolef especially depend heavily on the template support of the compiler. The C++ 2011 standard features are recommended but still optional: when available, it allows extended features, e.g. an elegant matrix concatenation syntax via the `std::initializer_list` class.

Current distributed version of Rheolef has been heavily tested with GNU C++, CLANG C++ and INTEL C++ on debian, ubuntu, mint, suse and redhat GNU/Linux systems for various 32 and 64 bits architectures, up to 200 distributed memories and processes. Successful installation from sources on MAC-OS-X systems has been also reported.

The GNU C++ compiler is a free software available at <http://www.gnu.org>. The CLANG C++ compiler is also a free software available at <http://clang.llvm.org>. The Intel C++ compiler is a privative software.

So, the Rheolef library is expected to compile without problems with one of these compilers. Otherwise, please send a bug report to <mailto:rheolef@grenet.fr>.

Older Rheolef 5.x versions was tested also on the KAI C++ compiler (KCC-3.2b) and the CRAY C++ compiler (CC-3.0.2.0) on the following operating systems:

```
mac os x (i386-apple-darwin9.8.0), using GNU C++
sun solaris 2.8, ultrasparc, using GNU C++
compacq OSF1 alpha (V5.1, V4.0), using Compacq C++
aix 4.1 ibm sp2, multiprocessor based on rs6000
cray unicos t3e, multiprocessor based on 64 bits dec alpha
cray unicos c94
hpux 9.05 and 10.20, hppa
sgi irix 6.3, mips
sun solaris 2.5.1, sparc
```

The KAI C++ is available at <http://www.kai.com>. The CRAY C++ is available on CRAY platforms. Nevertheless, the current version has no more been tested on these compilers and systems.

1.10 Configure command line options

If the required libraries are located in a non-standard directory, please use the configure options:

```
./configure --help
```

and also consider the CXX, CXXFLAGS and LIBS variables, as in the following more complex configure command:

```
CXX=clang++ CXXFLAGS=-I/misc/include LIBS="-L/misc/lib" ./configure
```

Finally, if troubles still persist after the configure step, you can also edit directly the `config/config.mk` and `config/config.h` files. In these cases please also post to the mailing list <mailto:rheolef@grenet.fr> for any portability problem.

If you are running a another compiler and operating system combination, please run the non-regression test suite:

```
make check
```

Please, send mails and bug reports to the mailing list <mailto:rheolef@grenet.fr>.

1.11 Configure command line options

`--with-float=option`

Turns on/off the extended arithmetic feature. The *option* could be either `float128` (quadruple precision, 32 digits), or the default `double` (double precision, 16 digits).

`--enable-permissive`

The compiler do not turns all warnings into errors and becomes permissive. This feature is for code developers and is off by default.

`--with-bigfloat=digits10`

Turn on Bruno Haible's `cln` arbitrary precision float library with *digits10* precision,

See <http://clisp.cons.org/~haible/packages-cln.html>. Default is off. The *digits10* value is optional and default *digits10* value is 60. This feature is still under development.

`--with-cln=dir_cln`

Set the home directory for the `cln` library. Default *dir_cln* is `'/usr/local/math'`.

`--enable-debian-packaging=debian_packaging`

Generate files the debian way by respecting the 'Debian Policy Manual'.

`--with-boost-incdir=incdir_boost`

`--with-boost-libdir=libdir_boost`

Specifies the `boost` library. This library is required by rheolef.

`--with-armadillo-incdir=incdir_armadillo`

`--with-armadillo-libdir=libdir_armadillo`

Specifies the `armadillo` library. This library is required by rheolef.

`--enable-debug`

Produce a `c++ -g` like code.

`--enable-dmalloc`

With debugging, also uses dynamic allocation runtime checking with `dmalloc` when available.

`--enable-old-code`
 Turns on/off the old code branch (before distributed computation features).
 Default is off.

`--enable-mpi`
 Turns on/off the distributed computation features. Default is on when distributed is on. This feature is currently in development. When on, configure try to detect either the scotch or the parmetis libraries.

`--with-scotch-incdir=incdir_scotch`
`--with-scotch-libdir=libdir_scotch`
`--with-scotch-libs=libs_scotch`
 Turns on/off the `scotch` distributed mesh partitioner. Check in `libdir_scotch` for `libptscotchparmetis.so`, `libptscotch.so` and `libptscotcherrexit.so` or corresponding `.a` libraries. Default is to auto-detect when mpi is available.

`--with-parmetis-incdir=incdir_parmetis`
`--with-parmetis-libdir=libdir_parmetis`
 Turns on/off the `parmetis` distributed mesh partitioner. Check in `libdir_parmetis` for `libparmetis.so`, `libparmetis.a` and also `libmetis.a` `libmetis.so`. Default is to autodetect when mpi is available and scotch unavailable.

`--with-blas-libdir=rheo_libdir_blas`
`--with-blas-libs=rheo_libs_blas`
 Turns on/off the `blas` libraries. Check in `rheo_libs_blas` for `libblas.so`, or the corresponding `.a` libraries. Default is to auto-detect. This library is required for the pastix library.

`--with-scalapack-libdir=rheo_libdir_scalapack`
`--with-scalapack-libs=rheo_libs_scalapack`
 Turns on/off the `scalapack` libraries. Check in `rheo_libs_scalapack` for `libscalapack.so`, or the corresponding `.a` libraries. Default is to auto-detect. This library is required for the pastix library.

`--with-mumps-incdir=incdir_mumps`
`--with-mumps-libdir=libdir_mumps`
`--with-mumps-libs=libs_mumps`
 Turns on/off the `mumps` distributed solver. Check in `libdir_mumps` for `libmumps.so`, or the corresponding `.a` libraries. Default is to auto-detect when mpi is available.

`--with-trilinos`
`--with-trilinos=libdir_trilinos`
 Turns on/off the `trilinos` distributed preconditioner library. Check in `libdir_trilinos` for `libtrilinos_ifpack.so`, or the corresponding `.a` libraries. Default is to auto-detect when mpi is available.

`--with-pastix`
`--with-pastix=libdir_pastix`
 Turns on/off the `pastix` distributed solver. Check in `libdir_pastix` for `libpastix.so`, or the corresponding `.a` libraries. Default is to auto-detect when mpi is available.

```

--with-cholmod-incdir=incdir_cholmod
--with-cholmod-libdir=libdir_cholmod
--with-cholmod-libs=libs_cholmod
    Turns on/off the cholmod multifrontal direct solver. Default incdir_cholmod is
    libdir_cholmod. Check in libdir_cholmod for libcholmod.so or libcholmod.a
    and for header incdir_cholmod/cholmod.h or incdir_cholmod/cholmod/cholmod.h.

--with-umfpack-incdir=incdir_umfpack
--with-umfpack-libdir=libdir_umfpack
--with-umfpack-libs=libs_umfpack
    Turns on/off the umfpack version 4.3 multifrontal direct solver. Default in-
cdir_umfpack is libdir_umfpack. Check in libdir_umfpack for libumfpack.so or
libumfpack.a and for header incdir_umfpack/umfpack.h or incdir_umfpack/umfpack/umfpack.h.

--with-taucs-ldadd=taucs_ldadd
--with-taucs-incdir=taucs_incdir
    Turns on/off the taucs version 2.0 out-of-core sparse direct solver. When this
    library is not available, the configure script check for the spooles multifrontal
    (see below).

--with-spooles-libdir=libdir_spooles
--with-spooles-incdir=incdir_spooles
    Turns on/off the spooles version 2.2 multifrontal direct solver. Default in-
cdir_spooles is libdir_spooles. Check in libdir_spooles for libspooles.so,
libspooles.a or spooles.a and for header incdir_spooles/FrontMtx.h. When
    this library is not available, the direct solver bases upon a traditionnal skyline
    Choleski factorisation combined with a reverse Cuthill-Mc Kee reordering.

--with-cgal-incdir=incdir_cgal
--with-cgal-libdir=libdir_cgal
    Turns on/off the cgal distributed solver. Check in libdir_cgal for libcgal.so,
    or the corresponding .a libraries. Default is to auto-detect. Generic or
    hardware-dependant optimization

--enable-optim
    Turns compile-time optimization to maximum available. Include architecture-
    dependent compilation (e.g. on intel, executable are less portable accross intel
    variants). Default is on. optimize or not ? architecture-dependent optim
    architecture-independent optim, when no debug rpath issue for debian package:
    seems to be buggy with DESTDIR!="

```

1.12 The directory structure

config	portability related files and documentation extractors
util	a collection of useful C++ classes and functions for smart pointers, handling files and directories, using strings, timing.
skit	sparse matrix library, linear solvers.
nfem	finite element library, mesh, spaces and forms.
doc	reference manual, user's guide and examples.

2 Reporting bugs

This software is still under development. Please, run the `make check` non-regression tests. Send comments and bug reports to `Pierre.Saramito@imag.fr`. Include the version number, which you can find at the top of this document. Also include in your message the input and the output that the program produced and some comments on the output you expected.

3 Commands

3.1 branch – handle a family of fields

(Source file: ‘nfem/pbin/branch.cc’)

Synopsis

```
branch [options] filename
```

Example

Generates vtk file collection for visualization with paraview:

```
branch output.branch -paraview
```

Description

Read and output a branch of finite element fields from file, in field text file format.

Input file specification

- Idir** add *dir* to the RHEOPATH search path. See also [\[geo class\]](#), page [\[undefined\]](#) for RHEOPATH mechanism.
- filename** specifies the name of the file containing the input field.
- read field on standard input instead on a file.
- ndigit int** Number of digits used to print floating point values when using the ‘-geo’ option. Default depends upon the machine precision associated to the **Float** type.

Inquire

- toc** print the table of contents (toc) to standard output and exit.

Output and render specification

- extract int** Extract the *i*-th record in the file. The output is a field or multi-field file format.
- branch** Output on stdout in ‘.branch’ format. This is the default.
- paraview** Generate a collection of **vtk** files for using **paraview**.
- vtk** Generate a single **vtk** file with numbered fields.
- gnuplot** Run 1d animation using **gnuplot**.
- plotmtv** This driver is unsupported for animations.

Other options

- umin** *float*
- umax** *float*
 set the solution range for the **gnuplot** driver. By default this range is computed from the first field of the branch, and this could be problematic when this field is initially zero.
- subdivide** *int*
 When using a high order geometry, the number of points per edge used to draw a curved element. Default value is the mesh order.
- topography** *filename[.field.gz]*
 performs a tridimensionnal elevation view based on the topographic data.
- proj** performs a P1 projection on the fly. This option is useful when rendering P0 data while **vtk** render requires P1 description.
- elevation**
 For two dimensional field, represent values as elevation in the third dimension. This is the default.
- noelevation**
 Prevent from the elevation representation.
- scale** *float*
 applies a multiplicative factor to the field. This is useful e.g. in conjunction with the **elevation** option. The default value is 1.
- verbose** print messages related to graphic files created and command system calls (this is the default).
- noverbose**
 does not print previous messages.
- clean** clear temporary graphic files (this is the default).
- noclean** does not clear temporary graphic files.
- execute** execute graphic command (this is the default).
- noexecute**
 does not execute graphic command. Generates only graphic files. This is useful in conjunction with the **-noclean** command.

Branch file format

The '**.branch**' file format bases on the '**.field**' one:

EXAMPLE	GENERAL FORM
#!branch	#!branch
branch	branch
1 1 11	<version> <nfield=1> <nvalue=N>
time u	<key> <field name>

```

#time 3.14      #<key> <key value 1>
#u              #<field name>
field           <field 1>
.....         ....

.....         ....
#time 6.28      #<key> <key value N>
#u              #<field name>
field           <field N>
.....         ....

```

The key name is here `time`, but could be any string (without spaces). The previous example contains one `field` at each time step. Labels appears all along the file to facilitate direct jumps and field and step skips.

The format supports several fields, such as $(t, u(t), p(t))$, where u could be a multi-component (e.g. a vector) field:

```

#!branch
branch
 1 2 11
time u p

#time 3.14
#u
mfield
1 2
#u0
field
...
#u1
field
...
#p

#time 6.28
...

```

3.2 field – plot a field

(Source file: ‘`nfem/pbin/field.cc`’)

Synopsis

```
field options filename[.field[.gz]]
```

Description

Read and output a finite element field from file.

Example

```
field square.field
field square.field -bw
field box.field
```

Input file specification

- filename* specifies the name of the file containing the input field.
- read field on standard input instead on a file.
- name when the field comes from standard input, the file base name is not known and is set to "output" by default. This option allows one to change this default. Useful when dealing with output formats (graphic, format conversion) that creates auxiliary files, based on this name.
- I*dir* Add *dir* to the rheolef file search path. This option is useful e.g. when the mesh .geo and the .field files are in different directories. This mechanism initializes a search path given by the environment variable 'RHEOPATH'. If the environment variable 'RHEOPATH' is not set, the default value is the current directory.
- mark *string*
- catch *string*
- catchmark *string* Jump across the file to the specifield *string*. Label start at the beginning of a line, preceded by a '#' mark (see see (undefined) [catchmark algorithm], page (undefined)).

Output file format options

- text
- field output field on standard output stream in Rheolef ascii (field or geo) text file format.
- gmsh output field on standard output stream in gmsh text file format.
- gmsh-pos output field on standard output stream in gmsh-pos text file format, suitable for mesh adaptation purpose.
- bamg-bb output field on standard output stream in bamg-bb text file format, suitable for mesh adaptation purpose.
- image-format *string* The argument is any valid image format, such as bitmap **png**, **jpg**, **gif**, **tif**, **ppm**, **bmp** or vectorial **pdf**, **eps**, **ps**, **svg** image file formats. this option can

be used with the `paraview` and the `gnuplot` renders. The output file is e.g. *basename.png* when *basename* is the name of the mesh, or can be set with the `-name` option.

`-resolution int int`

The argument is a couple of sizes, for the image resolution. This option can be used together with the `-image-format` for any of the bitmap image formats. This option requires the `paraview` render.

Getting information

`-min`

`-max` print the min (resp. max) value of the scalar field and then exit.

`-get-geo` print the name of the mesh associated to the field and exit.

Render options

`-gnuplot` use `gnuplot` tool. This is the default in one dimension.

`-paraview`

use `paraview` tool. This is the default for two- and tri-dimensional geometries.

`-mayavi` use `mayavi` tool. This is maintained for backward compatibility purpose: it has been superseded by the `paraview` render.

Rendering options

`-color`

`-gray`

`-black-and-white`

`-bw` Use (color/gray scale/black and white) rendering. Color rendering is the default.

`-label`

`-nolabel` Show or hide title, color bar and various anotations. Default is to show labels.

`-elevation`

`-noelevation`

For two dimensional field, represent values as elevation in the third dimension. The default is no evelation.

`-scale float`

applies a multiplicative factor to the field. This is useful e.g. in conjunction with the `elevation` option. The default value is 1.

`-stereo`

`-nostereo`

Rendering mode suitable for red-blue anaglyph 3D stereoscopic glasses. This option is only available with `mayavi` and `paraview`.

- `-fill` isoline intervals are filled with color. This is the default.
- `-nofill` draw isolines by using lines.
- `-volume`
- `-novolume` for 3D data, render values using a colored translucent volume. This option requires the `paraview` code.
- `-cut`
- `-nocut` Cut by a specified plane. The cutting plane is specified by its origin point and normal vector. This option requires the `paraview` code.
- `-origin float [float [float]]`
 set the origin of the cutting plane. Default is (0.5, 0.5, 0.5).
- `-normal float [float [float]]`
 set the normal of the cutting plane. Default is (1, 0, 0).
- `-iso [float]`
 do draw 3d isosurface. When the optional float is not provided, a median value is used. This option requires the `paraview` code.
- `-noiso` do not draw isosurface.
- `-n-iso int`
 For 2D visualizations, the isovalue table contains regularly spaced values from `fmin` to `fmax`, the bounds of the field.
- `-proj approx`
 Convert all selected fields to approximation *approx* by using a L2 projection.
- `-proj` Convert all selected fields to Pk-continuous approximation by using a L2 projection.
- `-lumped-proj`
 Force P1 approximation for L2 projection and use a lumped mass matrix for it.
- `-round [float]`
 Round the input up to the specified precision. This option, combined with `-field`, leads to a round filter. Useful for non-regression test purpose, in order to compare numerical results between files with a limited precision, since the full double precision is machine-dependent.
- `-n-iso-negative int`
 The isovalue table is split into negatives and positives values. Assume there is `n_iso=15` isolines: if 4 is requested by this option, then, there will be 4 negatives isolines, regularly spaced from `fmin` to 0 and `11=15-4` positive isolines, regularly spaced from 0 to `fmax`. This option is useful when plotting e.g. vorticity or stream functions, where the sign of the field is representative.
- `-subdivide int`
 When using a high order geometry, the number of points per edge used to draw a curved element. Default value is the mesh order.

-deformation

Render vector-valued fields as deformed mesh using `mayavi`, `paraview` or `gnuplot`. This is the default vector field representation.

-velocity

Render vector-valued fields as arrows using `mayavi` or `paraview`.

Component extraction and domain reduction**-comp *int*****-comp *string***

Extract the *i*-th component of a vector-valued field. For a tensor-valued field, indexing components as "00", "01", "11"... is supported.

-domain *name*

Reduce the visualization to the specified domain.

Others options

-verbose print messages related to graphic files created and command system calls (this is the default).

-noverbose

does not print previous messages.

-clean clear temporary graphic files (this is the default).

-noclean does not clear temporary graphic files.

-execute execute graphic command (this is the default).

-noexecute

does not execute graphic command. Generates only graphic files. This is useful in conjunction with the **-noclean** command.

Field file format

It contains a header and a list values at degrees of freedom. The header contains the **field** keyword followed by a line containing a format version number (presently 1), the number of degrees of freedom (i.e. the number of values listed), the mesh file name without the `‘.geo’` extension the approximation (e.g. P1, P2, etc), and finally the list of values: A sample field file (compatible with the sample mesh example presented in command manual; see `[geo command]`, page `[undefined]`) writes:

```
field
1 4
square
P1
0.0
1.0
2.0
3.0
```


Examples

```
field cube.field -cut -normal 0 1 0 -origin 0.5 0.5 0.5 -vtk
```

This command send to `vtk` the cutted 2d plane of the 3d field.

```
field cube.field -cut -normal 0 1 0 -origin 0.5 0.5 0.5 -text > cube-cut.field
```

This command generates the cutted 2d field and its associated mesh.

```
field cube.field -iso 0.5
```

This command draws the isosurface.

```
field cube.field -iso 0.5 -text > isosurf.geo
```

This command generates the isosurface as a 3d surface mesh in ‘.geo’ format. This is suitable for others treatments.

3.3 geo - plot a finite element mesh

(Source file: ‘`nfem/pbin/geo.cc`’)

Synopsis

```
geo options mesh[.geo[.gz]]
```

Description

Plot or upgrade a finite element mesh.

Examples

Plot a mesh:

```
geo square.geo
geo box.geo
geo box.geo -full
```

Plot a mesh into a file:

```
geo square.geo -image-format png
```

Convert from a old geo file format to the new one:

```
geo -upgrade - < square-old.geo > square.geo
```

See below for the geo file format specification. The old file format does not contains edges and faces connectivity in 3d geometries, or edges connectivity in 2d geometries. The converter add it automatically into the upgraded file format. Conversely, the old file format is useful when combined with a translator from another file format that do not provides edges and faces connectivity.

Input file specification

- filename* specifies the name of the file containing the input mesh. The ".geo" suffix extension is assumed.
- read mesh on standard input instead on a file.
- name when mesh comes from standard input, the mesh name is not known and is set to "output" by default. This option allows one to change this default. Useful when dealing with output formats (graphic, format conversion) that creates auxiliary files, based on this name.
- I *dir* Add *dir* to the rheolef file search path. This mechanism initializes a search path given by the environment variable 'RHEOPATH'. If the environment variable 'RHEOPATH' is not set, the default value is the current directory.
- check Check that element orientation is positive.

Input format options

- if *format*
- input-format *format* load mesh in *format* file format, instead of plotting it. Supported output formats are: **geo**, **bang**, **vtk**. When loading from a file, the corresponding suffix extension is assumed.

Render specification

- gnuplot Use gnuplot tool. This is the default for 1D and 2D geometries.
- paraview Use **paraview** tool. This is the default for 3D geometries.
- mayavi Use **mayavi** tool. This tool has been superseded by **paraview** for 3D geometries. It its maintained for backward compatibility purpose.

Render options

- [no]lattice When using a high order geometry, the lattice inside any element appears. Default is on;
- [no]full All internal edges appears, for 3d meshes. Default is off;
- [no]fill Fill mesh faces using light effects, when available.
- [no]stereo Rendering mode suitable for red-blue anaglyph 3D stereoscopic glasses. Option only available with **mayavi** or **paraview**.

- [no]shrink
shrink elements (with **paraview** only).
- [no]cut cut by plane and clip (with **mayavi** or **paraview** only).
- [no]label
Show or hide labels, boundary domains and various annotations. By default, domains are showed with a specific color.
- round [*float*]
Round the input up to the specified precision. This option, combined with **-geo**, leads to a round filter. Useful for non-regression test purpose, in order to compare numerical results between files with a limited precision, since the full double precision is machine-dependent.

Output file format options

- geo output mesh on standard output stream in geo text file format, instead of plotting it.
- upgrade Convert from a old geo file format to the new one.
- gmsb output mesh on standard output stream in gmsb text file format, instead of plotting it.
- image-format *string*
The argument is any valid image format, such as bitmap **png**, **jpg**, **gif**, **tif**, **ppm**, **bmp** or vectorial **pdf**, **eps**, **ps**, **svg** image file formats. this option can be used with the **paraview** and the **gnuplot** renders. The output file is e.g. *basename.png* when *basename* is the name of the mesh, or can be set with the **-name** option.
- resolution *int int*
The argument is a couple of sizes, for the image resolution. This option can be used together with the **-image-format** for any of the bitmap image formats. This option requires the **paraview** render.

Others options

- subdivide *int*
Subdivide each edge in *k* parts, where *k* is the prescribed argument. The new vertices are numbered so that they coincide with the P_k Lagrange nodes. It can be combined with the **-geo** option to get the subdivided mesh. In that case, default value is 1, i.e. no subdividing. It can also be combined with a graphic option, such that **-gnuplot** or **paraview**: When dealing with a curved high order geometry, *k* corresponds to the number of points per edge used to draw a curved element. In that case, this option is activated by default and value is the curved mesh order.
- add-boundary
check for a domain named "boundary"; If this domain does not exists, extract the boundary of the geometry and append it to the domain list. This command

is useful for mesh converted from generators, as **bamg**, that cannot have more than one domain specification per boundary edge.

- rz**
- zr** Specifies the coordinate system. Useful when converting from bamg or gmsh format,.
- verbose** print messages related to graphic files created and command system calls (this is the default).
- noverbose** does not print previous messages.
- clean** clear temporary graphic files (this is the default).
- noclean** does not clear temporary graphic files.
- execute** execute graphic command (this is the default).
- noexecute** does not execute graphic command. Generates only graphic files. This is useful in conjunction with the "-noclean" command.
- check**
- dump** used for debug purpose.

Inquire options

- size**
- n-element** print the mesh size, i.e. the number of elements and then exit.
- n-vertex** print the number of elements and then exit.
- sys-coord** print the coordinate system and then exit.
- hmin**
- hmax** print the smallest (resp. largest) edge length and then exit.
- xmin**
- xmax** print the bounding box lower-left (resp. top-right) corner and exit.
- min-element-measure**
- max-element-measure** print the smallest (resp. largest) element measure and then exit.

Geo file format

This is the default mesh file format. It contains two entities, namely a mesh and a list of domains. The mesh entity starts with the **mesh** keyword, that should be at the beginning of a line. It is followed by the geo format version number: the current mesh format version number is 4. Next comes the header, containing global information: the space dimension

(e.g. 1, 2 or 3), the number of nodes and the number of elements, for each type of element (tetrahedra, etc). When dimension is three, the number of faces (triangles, quadrangles) is specified, and then, when dimension is two or three, the number of edges is also specified. Follows the node coordinates list and the elements connectivity list. Each element starts with a letter indicating the element type:

'p'	point
'e'	edge
't'	triangle
'q'	quadrangle
'T'	tetrahedron
'P'	prism
'H'	hexaedron

Then, for each element, comes the vertex indexes. A vertex index is numbered in the C-style, i.e. the first index started at 0 and the larger one is the number of vertices minus one. A sample mesh writes:

```

mesh
4
header
  dimension 2
  nodes      4
  triangles  2
  edges      5
end header
0 0
1 0
1 1
0 1
t  0 1 3
t  1 2 3
e  0 1
e  1 2
e  2 3
e  3 0
e  1 3

```

Note that information about edges for 2d meshes and faces for 3d one are required for maintaining P2 and higher order approximation fields in a consistent way: degrees of freedom associated to sides requires that sides are well defined.

The second entity is a list of domains, that finishes with the end of file. A domain starts with the `domain` keyword, that should be at the beginning of a line. It is followed by the domain name, a simple string. Then, comes the domain format version: the current domain version number is 2. Next, the domain dimension and its number of elements. Finally, the list of elements: they are specified by the element index in the mesh, preceded by its orientation.

A minus sign specifies that the element (generally a side) has the opposite orientation, while the plus sign is omitted. A sample domain list writes:

```
domain
bottom
2 1 1
0
```

```
domain
top
2 1 1
2
```

Copy and paste the previous sample mesh data in a file, e.g. "square.geo". Be careful for the "mesh" and "domain" to be at the beginning of a line. Then enter:

```
geo square.geo
```

and the mesh is displayed.

File format conversion and simplified geo file format

Information about edges for 2d meshes and faces for 3d one is not provided by most mesh generators (e.g. gmsh or bamg). It could be complex to build this list, so a simplified file format is also supported, without faces and/or edges connectivity, and the geo command proposes to build it automatically and save it in a more complete, upgraded geo file.

The simplified version of the previous mesh is:

```
mesh
4
header
  dimension 2
  nodes      4
  triangles 2
end header
0 0
1 0
1 1
0 1
t 0 1 3
t 1 2 3
```

The domain list is no more able to refer to existing sides: edges are simply listed by their complete connectivity, thanks to the domain format version number 1. For the previous example, we have:

```
domain
bottom
1 1 1
e 0 1
```

```

domain
top
1 1 1
e 2 3

```

Copy and paste the previous simplified sample mesh data in a file, e.g. "square0.geo". Be careful for the "mesh" and "domain" to be at the beginning of a line. Then enter:

```
geo -upgrade -geo square0.geo
```

and the previous mesh with its complete connectivity is displayed: edges has been automatically identified and numbered, and domains now refers to edge indexes.

Note that, for the `gms` and `bamg` mesh generators, automatic file conversion is provided by the `msh2geo` and `bamg2geo` commands.

3.4 basis – view a basis

(Source file: 'nfem/gbasis/basis.cc')

Synopsis

```
basis name element options
```

Description

View a finite element polynomial basis on a given reference element (see [reference_element iclass](#), page [reference_element iclass](#)). Also, could view the node set used during interpolation.

Example

```

basis P5 t
basis P5[warburton] t
basis P5[warburton,monomial] t
basis P3 t -node
basis P5 -node-side 0
basis P3 T -node
basis B7 t
basis S7 t
basis RT0 t
basis RT3 t -node

```

Basis and element arguments

The basis *name* is specified as for the `space` and `basis` constructor argument (see [space class](#), page [space class](#) and [basis class](#), page [basis class](#)). The reference *element* is one of `e`, `t`, `q`, `T`, `P`, `H` (see [reference_element iclass](#), page [reference_element iclass](#), [edge iclass](#), page [edge iclass](#) [triangle iclass](#)),

page [\[quadrangle iclass\]](#), page [\[tetrahedron iclass\]](#), page [\[prism iclass\]](#), page [\[hexahedron iclass\]](#), page [\[gnuplot render\]](#). The `gnuplot` render is used for visualization.

Raw basis

The raw basis is used for computing the Vandermonde matrix and building the Lagrange basis. It can be also viewed by the `basis` unix command, suing the switches:

`-raw`

`-fem`

For instance:

```
basis M4 t
basis D5 e
basis B5 e -raw
```

Note that the `-raw` option is used to disambiguate between the Bernstein basis as a FEM basis and the Bernstein basis as a raw basis for building the Lagrange basis.

Visualization option

The `basis` command supports several modes of visualization:

- `-poly` Represents the polynomial functions in evelation, for 1D and 2D elements (3D elements visualization not yet supported). All basis polynomials are showed in an animation.
- `-node` Represents the node location, with distinct colors for each dimension associated to.
- `-side-node` Represents the node location, restricted on a specific side.

Others options

- `-verbose` print messages related to graphic files created and command system calls (this is the default).
- `-noverbose` does not print previous messages.
- `-clean` clear temporary graphic files (this is the default).
- `-noclean` does not clear temporary graphic files.
- `-execute` execute graphic command (this is the default).
- `-noexecute` does not execute graphic command. Generates only graphic files. This is usefull in conjunction with the `-noclean` command.

Limitations

Polynomial visualization in 3D are not yet supported: future development will use paraview with volume mode and animation for scanning all basis polynomials.

3.5 bamg2geo - convert bamg mesh in geo format

(Source file: 'nfem/sbin/bamg2geo.cc')

Synopsis

```
bamg2geo options input[.bamg] input[.dmn]
```

Description

Convert a bamg '.bamg' into '.geo' one. The output goes to standard output. The '.dmn' file specifies the domain names, since bamg mesh generator uses numbers as domain labels. When no files are provided, the standard input is used.

Example

```
bamg -g toto.bamgcad -o toto.bamg
bamg2geo toto.bamg toto.dmn > toto.geo
```

Bamg cad file

This file describe the boundary of the mesh geometry. A basic example writes (See bamg documentation for more);

```
MeshVersionFormatted
0
Dimension
2
Vertices
4
0 0 1
1 0 2
1 1 3
0 1 4
Edges
4
1 2 101
2 3 102
3 4 103
4 1 104
hVertices
0.1 0.1 0.1 0.1
```

Domain name file

This auxiliary ‘.dmn’ file defines the boundary domain names as used by Rheolef, since `bang` uses numeric labels for domains.

```
EdgeDomainNames
4
bottom
right
top
left
```

The domain name file can also specify additional vertices domain:

```
EdgeDomainNames
4
bottom
right
top
left
VerticeDomainNames
4
left_bottom
right_bottom
right_top
left_top
```

Vertice domain names are useful for some special boundary conditions.

Options

```
-cartesian
-rz
-zr          Specifies the coordinate system.
```

Limitation

Do not support yet 2d region domains, such as "east" and "west".

3.6 `mkgeo_ball` – build an unstructured mesh of an ellipsoid, in 2d or 3d

(Source file: ‘`nfem/sbin/mkgeo_ball`’)

Synopsis

```
mkgeo_ball options [n]
```

Example

The following command build a triangle based 2d unstructured mesh of the unit ball

```
mkgeo_ball -t 10 > ball-10.geo
geo -mayavi ball-10.geo
```

or in one command line:

```
mkgeo_ball -t 10 | geo -mayavi -
```

Description

This command is useful when testing programs on simple geometries. Invocation is similar to `mkgeo_grid` (see [\[mkgeo_grid command\]](#), page [\[mkgeo_grid command\]](#), page [\[mkgeo_grid command\]](#)). It calls `gmsh` as unstructured mesh generator. It avoid the preparation of an input file for a mesh generator. The optional *n* argument is an integer that specifies the subdivision in each direction. By default *n*=10. The mesh files goes on standard output.

Element type options

`-t` 2d mesh using triangles.
`-q` 2d mesh using quadrangles.
`-tq` 2d mesh using both triangles and quadrangles.

The mesh order

`-order int`
 The polynomial pproximation mesh order, for the curved boundary, as defined by `gmsh`. Default is order=1.

The geometry

The geometry can be ellipse/ellipsoid inscribed in the $[a,b] \times [c,d]$ rectangle or the $[a,b] \times [c,d] \times [f,g]$ parallelotope. By default $a=c=f=-1$ and $b=d=g=1$, thus, the unit balls are considered.

`-s` Only a surface mesh is generated: a curved line in 2d or a curved surface mesh in 3d. In 3d, supports both `'-t'` and `'-q'` options.

`-a float`
`-b float`
`-c float`
`-d float`
`-f float`
`-g float`

Boundary domains

The meshes defines a domain named `boundary` that groups all boundary sides.

Others options

-fix
-nofix By default, internal face and volume node from gmsh are recomputed, since they have incorrect coordinate that destroy the convergence properties of isoparametric high order elements (order ≥ 3). The **-nofix** option skip this correction: this option is available for test purpose.
-clean clear temporary files (this is the default).
-noclean does not clear temporary files.
-verbose
-noverbose print intermediate commands and information messages.

3.7 mkgeo_contraction – build an unstructured 2d mesh of an abrupt contraction

(Source file: ‘nfem/sbin/mkgeo_contraction’)

Synopsis

```
mkgeo_contraction options [nx [ny [nz]]]
```

Example

The following command build a triangle-based 2d unstructured mesh of the $[-10,0] \times [0,4]$ u $[0,10] \times [0,1]$ mesh

```
mkgeo_contraction > contraction.geo
geo contraction.geo
```

Description

This command is convenient for building a mesh for the abrupt contraction, as it is a very classical benchmark in complex fluid flow problems. It calls **bamg** unstructured mesh generator with anisotropy feature. The mesh files goes on ‘*name*.geo’ where *name* is the basename for the output (see option **-name** below). The three auxiliary files required for automatic mesh generation with Rheolef combined with **bamg** are also provided: ‘*name*.bamg’, ‘*name*.bamgcad’ and ‘*name*.dmn’.

The geometry

The geometry is $[-Lu,0] \times [0,d]$ u $[0,Ld] \times [0,1]$. By default $c=4$ is the contraction ratio and $Lu=Ld=10$ are the upstream and downstream pipe lengths.

-c float
-Lu float
-Ld float These options control the geometry parameters.

-cartesian
-zr
-rz These options control the geometry coordinate system. Default is cartesian.

The discretization

The optional *nx* and *ny* arguments are floats that specifies the subdivision in each direction. By default *nx*=1 and *ny*=*ny*. Changing the density applies the factor $1/n$ to all the mesh edges lengths.

-hmin float
 Controls the edge length at the re-entrant corner of the contraction. Default is *hmin*=0.1. Changing the density by the previous options applies the factor $1/n$ to *hmin* also.

Boundary domains

The boundary sides are represented by domains: **axis**, **wall**, **upstream** and **downstream**.

Others options

-name string
 Set the basename for the output files. By default, the basename is **contraction**.

-split
-nosplit Split each triangle in three subtriangles by inserting the barycenter as an additional node. This is useful for using the Scott and Vogelius incompressible mixed finite element P2-P1d for velocity-pressure approximation. Default is no split.

-clean
-noclean Clear temporary files (this is the default).

-verbose
-noverbose
 In verbose mode, print to stderr all subcommands and logs.

3.8 mkgeo_grid – build a strutured mesh of a parallelotope, in 1d, 2d or 3d

(Source file: 'nfem/sbin/mkgeo_grid')

Synopsis

```
mkgeo_grid options [nx [ny [nz]]]
```

Example

The following command build a triangular based 2d 10x10 grid of the unit square:

```
mkgeo_grid -t 10 > square-10.geo
geo square-10.geo
```

or in one command line:

```
mkgeo_grid -t 10 | geo -
```

Description

This command is useful when testing programs on simple geometries. It avoid the preparation of an input file for a mesh generator. The optional *nx*, *ny* and *nz* arguments are integer that specifies the subdivision in each direction. By default *nx*=10, *ny*=*nx* and *nz*=*ny*. The mesh files goes on standard output.

The command supports all the possible element types: edges, triangles, rectangles, tetraedra, prisms and hexahedra.

Element type options

- e 1d mesh using edges.
- t 2d mesh using triangles.
- q 2d mesh using quadrangles (rectangles).
- T 3d mesh using tetraedra.
- P 3d mesh using prisms.
- H 3d mesh using hexahedra.

The geometry

The geometry can be any $[a,b]$ segment, $[a,b] \times [c,d]$ rectangle or $[a,b] \times [c,d] \times [f,g]$ parallelotope. By default $a=c=f=0$ and $b=d=g=1$, thus, the unit boxes are considered. For instance, the following command meshes the $[-2,2] \times [-1.5, 1.5]$ rectangle:

```
mkgeo_grid -t 10 -a -2 -b 2 -c -1.5 -d 1.5 | geo -
```

- a *float*
- b *float*
- c *float*
- d *float*
- f *float*
- g *float*

Boundary domains

- sides
- nosides The boundary sides are representd by domains: *left*, *right*, *top*, *bottom*, *front* and *back*.

-boundary
-noboundary

This option defines a domain named **boundary** that groups all sides.

By default, both sides and the whole boundary are defined as domains:

```
mkgeo_grid -t 10 > square.geo
geo square.geo
mkgeo_grid -t 10 -nosides > square.geo
geo square.geo
mkgeo_grid -t 10 -noboundary > square.geo
geo square.geo
mkgeo_grid -t 10 -noboundary -nosides > square.geo
geo square.geo
```

Regions

-region
-noregion

The whole domain is split into two subdomains: **east** and **west**. Also, the separating domain is named **interface** in the mesh. This option is used for testing computations with subdomains (e.g. transmission problem; see the user manual).

```
mkgeo_grid -t 10 -region | geo -
```

Corners

-corner
-nocorner

The corners (four in 2D and eight in 3D) are defined as OD-domains. This could be useful for some special boundary conditions.

```
mkgeo_grid -t 10 -corner | geo -
mkgeo_grid -T 5 -corner | geo -
```

Coordinate system option

Most of rheolef codes are coordinate-system independent. The coordinate system is specified in the geometry file `‘.geo’`.

-zr

-rz the 2d mesh is axisymmetric: **zr** (resp. **rz**) stands when the symmetry is related to the first (resp. second) coordinate.

File format option

3.9 mkgeo_sector – build an unstructured 2d mesh of a sector

(Source file: ‘nfem/sbin/mkgeo_sector’)

Synopsis

```
mkgeo_sector options [n]
```

Example

The following command build 2d unstructured mesh of the triangular region delimited by the three points (0,0), (1,0), (1,1):

```
mkgeo_sector
  geo sector.geo
```

Description

This command is convenient for building a mesh for a sector, as it is a very classical benchmark in complex fluid flow problems. It calls **bamg** unstructured mesh generator. The mesh files goes on ‘*name.geo*’ where *name* is the basename for the output (see option **-name** below). The three auxiliary files required for automatic mesh generation with Rheolef combined with **bamg** are also provided: ‘*name.bamg*’, ‘*name.bamgcad*’ and ‘*name.dmn*’.

The geometry

The geometry is the triangular sector (0,0), (a,0), (a,b). By default a=b=1.

-a *float*

-b *float* These options control the geometry parameters.

The discretization

The optional *n* argument is an integer that specifies the subdivision in each direction. By default *n*=10.

Boundary domains

The boundary sides are represented by domains: **axis**, **boundary**, and **bisector**.

Others options

-name *string*

Set the basename for the output files. By default, the basename is **sector**.

-clean

-noclean clear temporary files (this is the default).

-verbose

-noverbose

In verbose mode, print to stderr all subcommands and logs.

3.10 mkgeo_ugrid – build an unstructured mesh of a parallelotope, in 1d, 2d or 3d

(Source file: ‘nfem/sbin/mkgeo_ugrid’)

Synopsis

```
mkgeo_ugrid options [n]
```

Example

The following command build a triangle based 2d unstructured mesh of the unit square:

```
mkgeo_ugrid -t 10 > square-10.geo
geo -mayavi square-10.geo
```

or in one command line:

```
mkgeo_ugrid -t 10 | geo -mayavi -
```

Description

This command is useful when testing programs on simple geometries. Invocation is similar to `mkgeo_grid` (see `<undefined> [mkgeo_grid command]`, page `<undefined>`). It calls `gmsh` as unstructured mesh generator. It avoid the preparation of an input file for a mesh generator. The optional `n` argument is an integer that specifies the subdivision in each direction. By default `n=10`. The mesh files goes on standard output.

The command supports all the possible element types: edges, triangles, rectangles, tetraedra, prisms and hexahedra. It supports also mixed 2D with triangles and quadrangles:

```
mkgeo_ugrid -tq 10 | geo -mayavi -
```

and mixed 3D with tetraedra, prisms and/or hexahedra:

```
mkgeo_ugrid -TP 10 | geo -mayavi -
mkgeo_ugrid -PH 10 | geo -mayavi -
mkgeo_ugrid -TPH 10 | geo -mayavi -
```

Element type options

- e 1d mesh using edges.
- t 2d mesh using triangles.
- q 2d mesh using quadrangles.
- tq 2d mesh using both triangles and quadrangles.
- T 3d mesh using tetraedra.
- P 3d mesh using prisms.
- H 3d mesh using hexahedra.
- TP
- PH
- TPH 3d mesh using a mixt between tetraedra, prisms and/or hexahedra.

The geometry

The geometry can be any $[a,b]$ segment, $[a,b] \times [c,d]$ rectangle or $[a,b] \times [c,d] \times [f,g]$ paralleloptope. By default $a=c=f=0$ and $b=d=g=1$, thus, the unit boxes are considered. For instance, the following command meshes the $[-2,2] \times [-1.5, 1.5]$ rectangle:

```
mkgeo_ugrid -t 10 -a -2 -b 2 -c -1.5 -d 1.5 | geo -
```

-a *float*

-b *float*

-c *float*

-d *float*

-f *float*

-g *float*

Boundary domains

-sides

-nosides The boundary sides are represented by domains: **left**, **right**, **top**, **bottom**, **front** and **back**.

-boundary

-noboundary

This option defines a domain named **boundary** that groups all sides.

By default, both sides and the whole boundary are defined as domains:

```
mkgeo_ugrid -t 10 > square.geo
geo square.geo
mkgeo_ugrid -t 10 -nosides > square.geo
geo square.geo
mkgeo_ugrid -t 10 -noboundary > square.geo
geo square.geo
mkgeo_ugrid -t 10 -noboundary -nosides > square.geo
geo square.geo
```

Regions

-region

-noregion

The whole domain is split into two subdomains: **east** and **west**, This option is used for testing computations with subdomains (e.g. transmission problem; see the user manual).

```
mkgeo_ugrid -t 10 -region | geo -
```

Corners

-corner

-nocorner

The corners (four in 2D and eight in 3D) are defined as OD-domains. This could be useful for some special boundary conditions.

```
mkgeo_ugrid -t 10 -corner | geo -
mkgeo_ugrid -T 5 -corner | geo -
```

The mesh order

-order int

The polynomial approximation mesh order, as defined by **gmsh**. This option enable a possible curved boundary, when applying a suitable nonlinear transformation to the mesh. Default is order=1.

Others options

-clean clear temporary files (this is the default).

-noclean does not clear temporary files.

3.11 msh2geo – msh2geo - convert gmsh mesh in geo format

(Source file: ‘nfem/sbin/msh2geo.cc’)

Synopsis

```
msh2geo [-zr|-rz] < input[.msh] > output.geo
msh2geo [-zr|-rz] input[.msh] > output.geo
```

Description

Convert a gmsh ‘.msh’ into ‘.geo’ one. The output goes to standard output. The input comes either from standard input or from a file.

Example

```
gmsh -2 toto.mshcad -o toto.msh
msh2geo < toto.msh > toto.geo
gmsh -2 -order 2 toto.mshcad -o toto2.msh
msh2geo < toto2.msh > toto2.geo
```

See the **gmsh** documentation for a detailed description of the ‘.mshcad’ input file for **gmsh**.

Coordinate system option

Most of rheolef codes are coordinate-system independent. The coordinate system is specified in the geometry file ‘.geo’.

-zr

-rz The 2d mesh is axisymmetric: **zr** (resp. **rz**) stands when the symmetry is related to the first (resp. second) coordinate.

-cartesian

The coordinate system is cartesian. This is the default.

Notes

Pk triangle, when $k \geq 5$, may have internal nodes renumbered: from the gmsh documentation:

The nodes of a curved element are numbered in the following order:

```
the element principal vertices;
the internal nodes for each edge;
the internal nodes for each face;
the volume internal nodes.
```

The numbering for face and volume internal nodes is recursive, i.e., the numbering follows that of the nodes of an embedded face/volume. The higher order nodes are assumed to be equispaced on the element.

In rheolef, internal triangle nodes are numbered from left to right and then from bottom to top. The numbering differ for triangle when $k \geq 5$. Thus, `msh2geo` fix the corresponding internal nodes numbering during the conversion.

Pk tetrahedrons and hexaedrons in gmsh and rheolef has not the same edge-node order and orientation. E.g. for tetrahedrons, edges 13 and 23 should be swapped and reoriented as 32 and 31. Thus, `msh2geo` fix the corresponding internal nodes numbering.

Todo

Fix for P3-tetra: swap edges orientations for 3,4,5 and swap faces 1 and 2. Check P4(T) for face orientation. Perform face visualisation with gnuplot face fill.

See also hexa edges orient and faces numbers and orient.

Check that node are numbered by vertex-node, then edge-node, then face(tri,qua)-node and then volume(T,P,H)-node. Otherwise, renumber all nodes.

Support for high order ≥ 6 element is not documented in gmsh, but gmsh supports it at run

3.12 bamg - bidimensional anisotropic mesh generator

(Source file: 'util/bamg/bamg-man.txt')

Synopsis

```
bamg options -g input[.bamgcad] -o output[.bamg]
```

Example

Generate the mesh of a square $]1,1[^2$ with a mesh size $h=0.666$ at all vertices. Enter the unix command:

```
bamg -g toto.bamgcad -o toto.bamg
```

The geometry is defined in the `square.bamgcad` file:

```

MeshVersionFormatted 0
Dimension 2
Vertices 4
-1 -1 1
 1 -1 2
 1  1 3
-1  1 4
Edges 4
1 2 1
2 3 2
3 4 3
4 1 4
hVertices
0.666 0.666 0.666 0.666

```

The file starts with vertices, coordinates and identifier. Then come the boundary edges, using vertices identifiers and defining a boundary edge identifier.

Outline

This software can

- 1) **create** a mesh from a geometry
- 2) **adapt** a mesh from an existing background mesh using a metric or a solution file.
- 3) **metric build**
just build a metric file, e.g. if you have another mesher .
- 3) **quality improve**
of an existing mesh, by generating a new mesh.
- 5) **interpolate**
a field from one mesh to another.

1) create

Create a mesh from a geometry. Example:

```
bamg -g toto.bamgcad -o toto.bamg
```

-g *filename*

the input file, specifying the geometry boundaries of the domain to mesh (bamg file format **DB mesh**).

-o *filename*

the output mesh file (bamg file format **DB mesh**). Some alternatives output file formats are supported with some **-oXY** options where **XY** is one of the supported output file formats (see below).

In addition, optional parameter can be added to specify a metric or the quality improvement. All the options are described below.

2) adapt

Adapt a mesh from a background mesh using a metric or solution file. Example:

```
bamg -b toto_bgd.bamg -Mbb toto_bgd_sol.bb -o toto_new.bamg
```

-b *filename*

the input background mesh, where the file suffixe defines the format of the file: `.amdba`, `.am_fmt`, `.am`, `.ftq`, `.nopo`. Otherwise the file is the bamg default **BD mesh** file format.

-Mbb *filename*

-MBB *filename*

-M *filename*

The input metric file. The **-Mbb** or **-MBB** specifies the solution file from which the metric is automatically computed, where the file is of type **bb** or **BB** (see file format below). An alternative is to specify directly the metric with the **-M** option (file format **Metric**).

-o *filename*

the output mesh file (bamg file format **DB mesh**). Some alternatives output file formats are supported with some **-oXY** options where **XY** is one of the supported output file formats (see below).

In addition, optional parameter can be added to control the metric generation and the quality improvement. All the options are described below.

3) metric build

Construct a metric file for an existing mesh and with a provided solution. This option can be used without generating a new mesh, e.g. if you have another mesher.

```
bamg -r toto_bgd.bamg -Mbb toto_bgd_sol.bb -oM toto_bgd.metric
```

-r *filename*

The input mesh file (bamg format **DB mesh**).

--Mbb *filename*

--MBB *filename*

The input provided solution, where the file is of type **bb** or **BB** (see file format below).

-oM *filename*

The output metric file, in file format **Metric** (see file format below).

In addition, optional parameter can be added to control the metric generation. All the options are described below.

4) quality improve

Improve quality for an existing mesh and generate a new mesh.

```
bamg -r toto_bgd.bamg -M toto_bgd.metric -o toto_new.bamg
```

-r *filename*

The input mesh file (bamg format **DB mesh**).

-M *filename*

The input metric file, in file format **Metric** (see file format below).

-o *filename*

the output mesh file (bamg file format **DB mesh**). Some alternatives output file formats are supported with some **-oXY** options where **XY** is one of the supported output file formats (see below).

In addition, optional parameter can be added to control the quality improvement. All the options are described below.

5) interpolate

In the adaption process, a solution has been computed with the background mesh. In order to transfer the solution of the problem under consideration on the new generated mesh, an interpolation of old solution is necessary. This tranferred solution may be a good initial guess for the solution on the new mesh. This interpolation is carried out in a P1 Lagrange context.

```
bamg -b toto_old.bamg -rbb toto_old.bb -r toto_new.bamg -obb toto_new.bb
```

-b *filename*

The destination input mesh file (bamg format **DB mesh**).

-rbb *filename*

-rBB *filename*

The origin input solution, where the file is of type **bb** or **BB** (see file format below).

-r *filename*

The origin input mesh file (bamg format **DB mesh**).

-wbb *filename*

-wBB *filename*

The output solution,as reinterpolated on the destination mesh.

Creation options

-hmax *float*

Set the value of the maximal edge size. Default value is the diameter of the domain to be meshed.

-hmin *float*

Set the value of the minimal edge size. Default value is related to the size of the domain to be meshed and the grid resolution used by the mesh generator (machine dependent).

-errg *float*

Set the value of the relative error on geometry of the boundary. Default value is 0.1. In any case this value is greater than $1/\sqrt{2}$. Remark that mesh size created by this option can be smaller than the **hmin** argument due to geometrical constraint.

-nbv *int*

Set the maximal number of vertices of the generated mesh. Default value is 50000.

Adaptation options

These options are relevant when computing a metric from a scalar field provided in a .bb file. Notice that, when providing a tensor metric in the .bb file, the metric computation is not performed and these options are not relevant.

-RelError

compute the metric with a relative error. This is the default. In this case, the metric field is defined by

$$M(x) = \frac{1}{\text{err} \cdot \text{coef}^2} \frac{|H(x)|}{\max(\text{CutOff}, |\eta(x)|)}$$

where **err**, **coef**, **CutOff** are adjustable parameters defined below, **eta** is the solution field read in the input file and **H** is its Hessian. Here $|\eta|$ denotes the absolute value of the field **eta** and $|H|$ is the tensor field composed of the absolute values of the Hessian eigenvalues and with the same eigenbasis as **H**.

-AbsError

compute the metric with an absolute error. In this case, the metric is defined by

$$M(x) = \frac{1}{\text{err} \cdot \text{coef}^2} \frac{|H(x)|}{(\sup(\eta) - \inf(\eta))}$$

where $\sup(\eta)$ and $\inf(\eta)$ denotes the two extremal values of the input solution field **eta**.

-coef *float*

the multiplicative coefficient on the mesh size. Default value is 1.0.

-err *float*

the level of the P1 interpolation error. Default value is 0.01. Recall that this error behaves as $O(h^2)$ locally, where **h** is the local mesh size. Remark on the two previous formulae that a change by a factor 1/4 is equivalent to a change by a factor 1/2 on the mesh size. So, either **coef** or **err** are specified in order to generate a convergent mesh family.

-CutOff *float*

the cut-off value used for the relative error criteria. Default value is $1e-5$.

-power *float*

Set the power parameter of hessian to construct the metric. Default value is 1.

- NbJacobi *int*
Set the number of iterations in a smoothing procedure during the metric construction. The 0 value implies no smoothing. Default value is 1.
- ratio *float*
Set the ratio for a prescribed smoothing on the metric. If ratio is 0 (default value) or less than 1.1, no smoothing on the metric is done. If ratio > 1.1 the speed of mesh size variation is bounded by log(ratio). Remark tht, as val is closer to 1, the number of vertices generated increases. This may be useful to control the thickness of refined regions near shocks or boundary layers.
- aniso
-iso The -aniso enforces the metric to be anisotropic. This is the default. Conversely, the metric may be of isotropic type with the -iso flag.
- anisomax *float*
Set the bound of mesh anisotropy with respect to minimal mesh size in all direction so the maximal mesh size in all direction is bounded by the ratio **anisomax**. The default value is 1e6. Remark that when **anisomax**=1, the generated mesh is isotropic.
- hminaniso *float*
Set the value of **hmin** the minimal edge size and set the aniso mode.
- maxsubdiv *float*
Change the metric such that the maximal subdivision of a background's edge is bound by the **maxsubdiv** number. The **maxsubdiv** number is always limited by 10 and this is the default value.
- KeepBackVertices
-noKeepBackVertices
Try to Keep old vertices (default). Otherwise, all vertices are created from scratch.
- NoRescaling
-Rescaling
Don't rescale the solution between [0,1] before metric computation Default is to rescale.

Quality improvement options

- NbSmooth *int*
Set the number of iterations of the mesh smoothing procedure. Default value is 3.
- omega *float*
Set the relaxation parameter of the smoothing procedure, Default value is 1.8.
- splitpbedge
-nosplitpbedge
Sometimes, an internal edge can have its two vertices on the boundary. This causes a triangle to have all its vertices on the boundary. With the

`-splitpbedge` option, this edge is split in two, and this situation is avoided. By default, don't split.

- `-thetaquad float`
to create quad with 2 triangles Merge two triangles into a quadrilateral when the four angles of the quadrilateral are in the range `[thetaquad, 180-thetaquad]`.
- `-2`
to create the mesh with a mesh size divided by two.
- `-2q`
to split all triangles in three quadrilaterals, and to split all quadrilaterals in four.

Output mesh format options

- `-o filename`
bamg DB mesh file format (default).
- `-oamdba filename`
amdba format.
- `-oftq filename`
ftq format.
- `-omsh filename`
msh format (freefem3 format).
- `-oam_fmt filename`
am_fmt format.
- `-oam filename`
am format.
- `-onopo filename`
nopo format.

Others options

- `-thetamax float`
Set the angular limit for a corner in degree to be curved. The angle is defined from two normals of two consecutive edges. The default is 180 degree, i.e. no corners are curved. This option is useful when no geometry are provided, e.g. remeshing from an other mesh file format (`am_fmt`, `amdba`, `nopo`, etc). This parameter is normally specified in the geometry boundaries file (in BD file format) by the `AngleOfCornerBound` optional section: when this file format is used, this option has no effect.
- `-v int`
Set the level of printing (verbosity), which can be chosen between 0 and 10. Default value is 1.

Geometry file format (bamgcad)

The general structure allows one to specify a mesh describing the geometry of the given domain. The identification of the boundaries are used to define boundary conditions for a partial derivative equation problem. In this case, some of the above sections are not relevant. First the required sections are:

```

MeshVersionFormatted 0
Dimension 2
Vertices nv
{xk yk ik} k=1:nv
Edges ne
{il jl kl} l=1:ne

```

Next, the optional sections:

```

SubDomain nd
{2 iek orientk idk} k=1:nd

```

A sub-domain, i.e. a bounded connex components of the plan is defined using one edge identifier *ie* along with an orientation information *orient*, indicating on which side of this entity the sub-domain lies. This feature is useful, e.g. when dealing with a domain with holes. The sub-domain number is *id*. If no sub-domain are defined, then we suppose to mesh all the bounded connex component of the plan. Remark: `SubDomainFromGeom` is equivalent to `SubDomain`.

```

AngleOfCornerBound angle

```

The `AngleOfCornerBound` specifies the angular limit for a corner in degree to be curved. The angle is defined from two normals of two consecutive edges. The default is 180 degree, i.e. no corners are curved. When this angle is defined, some corners could be specified not to be curved by

```

Corners nc
{ik} k=1:nc

```

The curved geometric representation of a boundary in two dimensions uses the edges provided in the data structure so as to define some curves of order three in the following way:

- * an edge whose endpoints are corners and if no additional information are provided will be represented by a straight segment,
- * an edge whose endpoints are corners but whose tangent is provided at one endpoint will be represented by a curve of degree two,
- * an edge whose endpoints are corners but whose tangents are provided at these corners will be represented by a curve of degree three,
- * an edge whose endpoints are not corners and with no additional information will be represented by a curve of degree three. Indeed, we use in this case the adjacent edges so as to evaluate the tangents at the edge endpoints.

In short, an edge defined by two information will be approached by a straight line, three information allow one to obtain a curve of degree two and four data, a curve of degree three. The tangents are optionally specified by:

```

TangentAtEdges nt
{iek ivek xt yt} k=1:nt

```

For the edge identifier *ie*, the tangent at its *ive* vertex (*ive* takes value 1 or 2) is specified by its components *xt* and *yt*. Giving the tangent vector of an edge by means of the tangent vector at a point enables us to deal with the case where several edges (boundary lines) are emanating from a point.

The required vertices, are the vertices of the support that must be present in the mesh as element vertices. Similarly, some edges can be required:

```
RequiredVertices nrv
{iv_k} k=1:nrv
RequiredEdges (nre
{ie_k} k=1:nre
```

The following features are planned for future work. For periodic boundary conditions, the section **EquivalencedEdges** indicates that two edges must be meshed the same way:

```
EquivalencedEdges nee
{ie1_k ie2_k} k=1:nee
```

Crack definition is the purpose of the **CrackedEdges** section. We specify then that an edge is identical in terms of geometry to another edge:

```
CrackedEdges nce
{ie1_k ie2_k} k=1:nce
```

More reading

The original site of the bamg mesh generator is <http://www.ann.jussieu.fr/hecht/ftp/bamg>. Please read <http://www.ann.jussieu.fr/hecht/ftp/bamg/bamg.pdf> for the detailed file formats and more advanced examples, e.g. a mesh adaptation loop to minimize the P1 Lagrange interpolation error.

Credits

Frederic Hecht <Frederic.Hecht@inria.fr> is the author of bamg. Pierre Saramito <Pierre.Saramito@imag.fr> writes this unix man page.

Copyright

Copyright (C) 1998-2018 Frederic Hecht <Frederic.Hecht@inria.fr> LGPLv3+: GNU LGPL version 3 or later <<http://gnu.org/licenses/lgpl.html>>. This is free software: you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law.

3.13 rheolef-config – get installation directories

(Source file: ‘rheolef-config.in’)

Example

The following command returns the rheolef libraries directory:

```
rheolef-config --libdir
```

An environment sanity check writes:

```
rheolef-config --check
```

Description

This command is useful when linking executables with rheolef: libraries locations are required by the link editor. Such directories are defined while configuring rheolef, before to compile and install see [\[Installing\]](#), page [\[Installing\]](#). The `rheolef-config` command returns these settings.

Note that `rheolef-config` could be used in Makefiles for the determination of linker flags.

Another useful feature is the `--check` option. When `rheolef` is installed in a user directory, i.e. not as root, the sane run-time environment depends upon two environment variables. The first one is the `PATH`: `bkindir` directory may be present in `PATH`. The second environment variable is related to shared libraries, and its name is system-dependent, e.g. `LD_LIBRARY_PATH` on most platforms and `SHLIB_PATH` on HP-UX. Its content may contains `bindir`.

```
rheolef-config --shlibpath-var
```

Since it is a common mistake to have incorrect values for these variable, for novice users or for adanced ones, especaly when dealing with several installed versions, the environment sanity check writes:

```
rheolef-config --check
```

If there is mistakes, a hint is suggested to fix it and the return status is 1. Instead, the return status is 0.

File options

`--version`

rheolef version.

`--help` print option summary and exit.

`--prefix` install architecture-independent files location.

`--exec-prefix`

architecture-dependent files location.

`--includedir`

include header directory.

`--bindir` executables directory.

`--mandir` man documentation directory.

`--libdir` object code libraries directory.

`--datadir`
`--datarootdir`
 read-only architecture-independent data location.

`--pkgdatadir`
 read-only architecture-independent data location; specific for package.

`--includes`
 include compiler flags.

`--libs` library compiler flags.

`--shlibpath-var`
 the shared library path variable.

`--library-interface-version`
 the library interface version.

`--hardcode-libdir-flag-spec`
 flag to hardcode a libdir into a binary during linking.

`--is-distributed`
 true or false: whether it is the distributed version.

`--float` returns the rheolef Float type. Float is **double** by default. It could also be an extended arithmetic precision type as **dd_real** (quadruple precision) or **qd_real** (octuple precision), depending on the configure option at compile time.

`--have-old-code`
`--have-new-code`
 true or false: whether it is the new/old code branch that is installed.

4 Classes

4.1 band - compute the band around a level set

(Source file: 'nfem/plib/band.h')

Description

Given a function `fh` defined in a domain `Lambda`, compute the band of elements intersecting the level set defined by $\{x \text{ in } \Lambda, fh(x) = 0\}$. This class is used for solving problems defined on a surface described by a level set function (See [\[level_set algorithm\]](#), page [\[undefined\]](#)).

Accessors

Each side in the surface mesh, as returned by the `level_set` member function, is included into an element of the band mesh, as returned by the `band` member function. Moreover, in the distributed memory environment, this correspondence is on the same process, so local indexes can be used for this correspondence: this is the `sid_ie2bnd_ie` member functions.

Band topology and domains

For the direct resolution of systems posed on the band, the mesh returned by the `band()` provides some domains of vertices. The "zero" vertex domain lists all vertices `xi` such that $fh(xi)=0$. The "isolated" vertex domain lists all vertices `xi` such that $fh(xi) \neq 0$ and `xi` is contained by only one element `K` and all vertices `xj!=xi` of `K` satisfies $fh(xj)=0$. Others vertices of the band, separated by the zero and isolated ones, are organized by connected components: the `n_connex_component` member function returns its number. Corresponding vertex domains of the band are named "cc<i>" where <i> should be replaced by any number between 0 and `n_connex_component-1`.

Implementation

```
template <class T, class M = rheo_default_memory_model>
class band_basic {
public:

    typedef typename geo_basic<T,M>::size_type size_type;

    // allocators:

    band_basic();
    band_basic(const field_basic<T,M>& fh,
               const level_set_option& opt = level_set_option());

    /// accessors:

    const geo_basic<T,M>& band() const { return _band; }
```



```

const geo_basic<T,M>& level_set() const { return _gamma; }
size_type sid_ie2bnd_ie (size_type sid_ie) const { return _sid_ie2bnd_ie [sid_ie]
size_type n_connected_component() const { return _ncc; }

// data:
protected:
    geo_basic<T,M>          _gamma;
    geo_basic<T,M>          _band;
    disarray<size_type,M>   _sid_ie2bnd_ie;
    size_type               _ncc;
};
typedef band_basic<Float> band;

```

4.2 branch - a parameter-dependent sequence of field

(Source file: 'nfem/plib/branch.h')

Description

Stores a field sequence together with its associated parameter value: a **branch** variable represents a pair $(t, u_h(t))$ for a specific value of the parameter t . Applications concern time-dependent problems and continuation methods.

This class is convenient for file inputs/outputs and building graphical animations.

Examples

Coming soon...

Limitations

This class is under development.

The **branch** class store pointers on field class without reference counting. Thus, **branch** automatic variables cannot be returned by functions. **branch** variable are limited to local variables.

Implementation

```

template <class T, class M = rheo_default_memory_model>
class branch_basic : public std::vector<std::pair<std::string,field_basic<T,M> > >
public :
// typedefs:

    typedef std::vector<std::pair<std::string,field_basic<T,M> > >    base;
    typedef typename base::size_type                                size_type;

// allocators:

    branch_basic ();

```

```

    ~branch_basic();

#ifdef _RHEOLEF_HAVE_VARIADIC_TEMPLATE
    template <typename... Args>
    branch_basic(const std::string& parameter, Args... names);
#else // ! _RHEOLEF_HAVE_VARIADIC_TEMPLATE
    branch_basic (const std::string& parameter_name, const std::string& u0);
    branch_basic (const std::string& parameter_name, const std::string& u0,
                                                         const std::string& u1);
    branch_basic (const std::string& parameter_name, const std::string& u0,
                                                         const std::string& u1,
                                                         const std::string& u2);
#endif // ! _RHEOLEF_HAVE_VARIADIC_TEMPLATE

// accessors:

    const T& parameter () const;
    const std::string& parameter_name () const;
    size_type      n_value () const;
    size_type      n_field () const;

// modifiers:

    void set_parameter_name (const std::string& name);
    void set_parameter (const T& value);
    void set_range (const std::pair<T,T>& u_range);

// input/output:

    // get/set current value
#ifdef _RHEOLEF_HAVE_VARIADIC_TEMPLATE
    template <typename... Args>
    __branch<T,M> operator() (const T& t, const field_basic<T,M>& u0, Args... uk);
    template <typename... Args>
    __iobranch<T,M> operator() (T& t,                                     field_basic<T,M>& u0, Args&... uk)
#else // ! _RHEOLEF_HAVE_VARIADIC_TEMPLATE
    __obranch<T,M> operator() (const T& t, const field_basic<T,M>& u0);
    __obranch<T,M> operator() (const T& t, const field_basic<T,M>& u0,
                                                         const field_basic<T,M>& u1);
    __obranch<T,M> operator() (const T& t, const field_basic<T,M>& u0,
                                                         const field_basic<T,M>& u1,
                                                         const field_basic<T,M>& u2);
    __iobranch<T,M> operator() (T& t, field_basic<T,M>& u0);
    __iobranch<T,M> operator() (T& t, field_basic<T,M>& u0,
                                                         field_basic<T,M>& u1);
    __iobranch<T,M> operator() (T& t, field_basic<T,M>& u0,
                                                         field_basic<T,M>& u1,
                                                         field_basic<T,M>& u2);
#endif // ! _RHEOLEF_HAVE_VARIADIC_TEMPLATE

```

```

__branch_header<T,M>      header ();
__const_branch_header<T,M> header () const;
__const_branch_finalize<T,M> finalize () const;

```

4.3 characteristic - the Lagrange-Galerkin method implemented

(Source file: 'nfem/plib/characteristic.h')

Synopsis

The class `characteristic` implements the Lagrange-Galerkin method: It is the extension of the method of characteristic from the finite difference to the finite element context.

Example

Consider the bilinear form `lh` defined by

$$lh(x) = \frac{\int_{\Omega} u_h(x+dh(x)) v(x) dx}{\int_{\Omega} 1 dx}$$

where `dh` is a deformation vector field. The characteristic is defined by $X(x)=x+dh(x)$ and the previous integral writes equivalently:

$$lh(x) = \frac{\int_{\Omega} u_h(X(x)) v(x) dx}{\int_{\Omega} 1 dx}$$

For instance, in Lagrange-Galerkin methods, the deformation field $dh(x)=-dt*uh(x)$ where `uh` is the advection field and `dt` a time step. The following code implements the computation of `lh`:

```

field dh = ...;
field uh = ...;
characteristic X (dh);
test v (Xh);
field lh = integrate (compose(uh, X)*v, qopt);

```

The Gauss-Lobatto quadrature formule is recommended for Lagrange-Galerkin methods. The order equal to the polynomial order of `Xh` (order 1: trapeze, order 2: simpson, etc). Recall that this choice of quadrature formulae guaranties unconditional stability at any polynomial order. Alternative quadrature formulae or order can be used by using the additional quadrature option argument to the `integrate` function see [\[integrate algorithm\]](#), page [\[undefined\]](#).

Implementation

```
template<class T, class M = rheo_default_memory_model>
class characteristic_basic : public smart_pointer<characteristic_rep<T,M> > {
public:
    typedef characteristic_rep<T,M> rep;
    typedef smart_pointer<rep> base;

    // allocator:

    characteristic_basic(const field_basic<T,M>& dh);

    // accesors:

    const field_basic<T,M>& get_displacement() const;

    const characteristic_on_quadrature<T,M>&
    get_pre_computed (
        const space_basic<T,M>&      Xh,
        const field_basic<T,M>&      dh,
        const quadrature_option& qopt) const;

};
typedef characteristic_basic<Float> characteristic;
```

4.4 continuation_option - send options to the continuation algorithm

(Source file: 'nfem/plib/continuation_option.h')

Description

This class is used to send options to the continuation function (see [\[continuation algorithm\]](#), page [\[undefined\]](#)).

Flags

tol

max_iter These options are transmitted to the `damped_newton` algorithm called during the continuation algorithm.

ini_delta_parameter

max_delta_parameter

max_delta_parameter

These options control the continuation parameter evolution along the branch of solution. When using the Keller continuation, these options control the arc-length parameter.

n_adapt When non-zero allows one to activate the optional mesh adaptation feature (see [\[adapt algorithm\]](#), page [\[undefined\]](#)) embedded in the continuation algorithm.

Todo

Complete this documentation.

Implementation

```
struct continuation_option : adapt_option {
    Float ini_direction;
    Float kappa;
    Float tol;
    size_t max_iter;
    size_t newton_max_iter;
    Float min_delta_parameter;
    Float max_delta_parameter;
    Float ini_delta_parameter;
    Float theta_decr;
    Float theta_incr;
    Float theta_variation;
    size_t min_delta_parameter_successive_count_max;
    Float tol_cos_angle;
    bool do_prediction;
    bool do_check_going_back;
    size_t n_adapt;
    continuation_option(const adapt_option& aopt = adapt_option());
    void check() const;
};
```

4.5 field - piecewise polynomial finite element field

(Source file: 'nfem/plib/field.h')

Description

Store degrees of freedom associated to a mesh and a piecewise polynomial approximation, with respect to the numbering defined by the underlying [\[space class\]](#), page [\[undefined\]](#).

This class contains two vectors, namely unknown and blocked degrees of freedoms, and the associated finite element space. Blocked and unknown degrees of freedom can be set by using domain name indexation:

```
geo omega ("circle");
space Xh (omega, "P1");
Xh.block ("boundary");
field uh (Xh);
uh ["boundary"] = 0;
```

Interpolation

Interpolation of a function `u` in a field `uh` with respect to the interpolation writes:

```
Float u (const point& x) { return x[0]*x[1]; }
...
field uh = interpolate (Xh, u);
```

Linear algebra

Linear algebra, such as `uh+vh`, `uh-vh` and `lambda*uh + mu*vh`, where `lambda` and `mu` are of type `Float`, are supported. The duality product between two fields `lh` and `vh` writes simply `dual(lh,vh)`: for discrete fields, it corresponds to a simple Euclidian dot product in \mathbb{R}^n . The application of a bilinear form (see `<undefined>` [form class], page `<undefined>`) writes `m(uh,vh)` and is equivalent to `dual(m*uh,vh)`.

Non-linear algebra

Non-linear operations, such as `sqrt(uh)` or `1/uh` are also available. Notice that non-linear operations do not returns in general piecewise polynomials: the value returned by `sqrt(uh)` may be filtered by `interpolate`,

```
field vh = interpolate (Xh, sqrt(uh));
```

the Lagrange interpolant, to becomes a piecewise polynomial: All standard unary and binary math functions `abs`, `cos`, `sin...` are available on fields. Also `sqr(uh)`, the square of a field, and `min(uh,vh)`, `max(uh,vh)` are provided. Binary functions can be used also with a scalar, as in

```
field vh = interpolate (Xh, max (abs(uh), 0));
field wh = interpolate (Xh, pow (abs(uh), 1./3));
```

For applying a user-provided function to a field, use the `compose` function:

```
field vh = interpolate(Xh, compose(f, uh));
field wh = interpolate(Xh, compose(f, uh, vh));
```

The composition supports also general unary and binary class-functions. Also, the multiplication `uh*vh` and the division `uh/vh` returns a result that is not in the same discrete finite element space: its result may be filtered by the `interpolate` operator:

```
field wh = interpolate(Xh, uh*vh);
```

Any function or class function can be used in nonlinear expressions: the function is interpolated in the specified finite element space.

There is a special predefined class-function named `normal` that represents the outer ununit normal vector on a boundary domain or surfacic mesh:

```
size_t k = omega.order();
string n_approx = "P" + itos(k-1) + "d";
space Nh (omega["boundary"], n_approx, "vector");
field nh = interpolate(Nh, normal());
```

The `normal()` function could appear in any nonlinear field expression: it is evaluated on the fly, based on the current mesh. Notice that when using isoparametric elements, the normal vector is no more constant along any face of the mesh. Also, on general curved domains, the unit normal vector is discontinuous across boundary element interfaces.

Access by domain

The restriction of a field to a geometric domain, says `"boundary"` writes `uh["boundary"]`: it represents the trace of the field on the boundary:

```
space Xh (omega, "P1");
uh["boundary"] = 0;
```

Extraction of the trace as a field is also possible:

```
field wh = uh["boundary"];
```

The space associated to the trace writes `wh.get_space()` and is equivalent to `space(omega["boundary"], "P1")`. See see `<undefined> [space class]`, page `<undefined>`.

Vector valued field

A vector-valued field contains several components, as:

```
space Xh (omega, "P2", "vector");
field uh (Xh);
field vh = uh[0] - uh[1];
field nh = norm (uh);
```

The `norm` function returns the euclidian norm of the vector-valuated field at each degree of freedom: its result is a scalar field.

Tensor valued field

A tensor-valued field can be constructed and used as:

```
space Th (omega, "P1d", "tensor");
field sigma_h (Xh);
field trace_h = sigma(0,0) + sigma_h(1,1);
field nh = norm (sigma_h);
```

The `norm` function returns the euclidian norm of the tensor-valuated field at each degree of freedom: its result is a scalar field. Notice that, as tensor-valued fields are symmetric, extra-diagonals are counted twice.

General multi-component interface

A general multi-component field writes:

```
space Th (omega, "P1d", "tensor");
space Vh (omega, "P2", "vector");
space Qh (omega, "P1");
```

```

space Xh = Th*Vh*Qh;
field xh (Xh);
field tau_h = xh[0]; // tensor-valued
field uh    = xh[1]; // vector-valued
field qh    = xh[2]; // scalar

```

Remark the hierarchical multi-component field structure: the first-component is tensor-valued and the second-one is vector-valued. There is no limitation upon the hierarchical number of levels in use.

For any field `xh`, the string `xh.valued()` returns `"scalar"` for a scalar field and `"vector"` for a vector-valued one. Other possible values are `"tensor"` and `"other"`. The `xh.size()` returns the number of field components. When the field is scalar, it returns zero by convention, and `xh[0]` is undefined. A vector-valued field has `d` components, where `d=omega.dimension()`. A tensor-valued field has `d*(d+1)/2` components, where `d=omega.dimension()`.

Blocked and unblocked arrays

The field class contains two vectors of degrees-of-freedom (dof) associated respectively to blocked and unknown dofs. Blocked dofs corresponds to Dirichlet boundary conditions, as specified by space (See [\[space class\]](#), page [\[undefined\]](#)). Access to these vectors is allowed via some accessors: a read-only one, as `uh.u()` and `uh.b()`, and a read-and-write one, as `uh.set_u()` and `uh.set_b()`, see [\[vec class\]](#), page [\[undefined\]](#).

Low-level degree-of-freedom access

The field class provides a STL-like container interface for accessing the degrees-of-freedom (dof) of a finite element field `uh`. The number of dofs is `uh.ndof()` and any dof can be accessed via `uh.dof(idof)`. A non-local dof at the partition interface can be obtained via `uh.dis_dof(dis_idof)` where `dis_idof` is the (global) distributed index associated to the distribution `uh.ownership()`.

For performances, a STL-like iterator interface is available, with `uh.begin_dof()` and `uh.end_dof()` returns iterators to the dofs on the current processor. See [\[disarray class\]](#), page [\[undefined\]](#) for more about distributed arrays.

For convenience, `uh.max()`, `uh.min()` and `uh.max_abs()` returns respectively the maximum, minimum and maximum of the absolute value of the degrees of freedom.

File format

TODO

Implementation note

The field expression uses the expression template technique in order to avoid temporaries when evaluating complex expressions.

Implementation

```

template <class T, class M = rheo_default_memory_model>
class field_basic : public std::unary_function<point_basic<typename scalar_traits<T>::value_type>,
public :
// typedefs:

    typedef typename std::size_t          size_type;
    typedef M                             memory_type;
    typedef T                             scalar_type;
    typedef typename float_traits<T>::type float_type;
// typedef undetermined_basic<T>          value_type; // TODO
    typedef T                             value_type; // TO_CLEAN
    typedef space_constant::valued_type    valued_type;
    typedef geo_basic <float_type,M>        geo_type;
    typedef space_basic<float_type,M>       space_type;
    typedef typename vec<T,M>::dis_reference dis_reference;
    class iterator;
    class const_iterator;

// allocator/deallocator:

    field_basic();

    explicit field_basic (
        const space_type& V,
        const T& init_value = std::numeric_limits<T>::max());

    void resize (
        const space_type& V,
        const T& init_value = std::numeric_limits<T>::max());

    field_basic<T,M>& operator= (const T&);
    field_basic<T,M>& operator= (const field_basic<T,M>&);

// linear expressions:
template <class Expr, class Sfinae
    = typename std::enable_if<
        details::is_field_expr_v2_linear_arg<Expr>::value
        && ! details::is_field<Expr>::value
    >::type>
field_basic (const Expr& expr);

template <class Expr, class Sfinae
    = typename std::enable_if<
        details::is_field_expr_v2_linear_arg<Expr>::value
        && ! details::is_field<Expr>::value
    >::type>
field_basic<T, M>& operator= (const Expr&);

```

```

// initializer list (c++ 2011):

#ifdef _RHEOLEF_HAVE_STD_INITIALIZER_LIST
    field_basic (const std::initializer_list<field_concat_value<T,M> >& init_list);
    field_basic<T,M>& operator= (const std::initializer_list<field_concat_value<T,M> >& init_list);
#endif // _RHEOLEF_HAVE_STD_INITIALIZER_LIST

// accessors:

    const space_type& get_space() const { return _V; }
    const geo_type& get_geo() const { return _V.get_geo(); }
    std::string stamp() const { return _V.stamp(); }
    std::string get_approx() const { return _V.get_approx(); }
    valued_type valued_tag() const { return _V.valued_tag(); }
    const std::string& valued() const { return _V.valued(); }

// accessors & modifiers to unknown & blocked parts:

    const vec<T,M>& u() const { return _u; }
    const vec<T,M>& b() const { return _b; }
    vec<T,M>& set_u() { dis_dof_indexes_requires_update(); return _u; }
    vec<T,M>& set_b() { dis_dof_indexes_requires_update(); return _b; }

// accessors to extremas:

    T min() const;
    T max() const;
    T max_abs() const;
    T min_abs() const;

// accessors by domains:

    field_indirect<T,M> operator[] (const geo_basic<T,M>& dom);
    field_indirect_const<T,M> operator[] (const geo_basic<T,M>& dom) const;
    field_indirect<T,M> operator[] (std::string dom_name);
    field_indirect_const<T,M> operator[] (std::string dom_name) const;

// accessors by components:

    size_type size() const { return _V.size(); }
    field_component<T,M> operator[] (size_type i_comp);
    field_component_const<T,M> operator[] (size_type i_comp) const;
    field_component<T,M> operator() (size_type i_comp, size_type j_comp);
    field_component_const<T,M> operator() (size_type i_comp, size_type j_comp) const;

// accessors by degrees-of-freedom (dof):

    const distributor& ownership() const { return get_space().ownership(); }

```

```

const communicator& comm() const { return ownership().comm(); }
size_type      ndof() const { return ownership().size(); }
size_type dis_ndof() const { return ownership().dis_size(); }
    T& dof (size_type idof);
const T& dof (size_type idof) const;
const T& dis_dof (size_type dis_idof) const;
// write access to non-local dis_idof changes to others procs
dis_reference dis_dof_entry (size_type dis_idof);

iterator begin_dof();
iterator end_dof();
const_iterator begin_dof() const;
const_iterator end_dof() const;

// input/output:

    idiststream& get (idiststream& ips);
    odiststream& put (odiststream& ops) const;
    odiststream& put_field (odiststream& ops) const;

// evaluate uh(x) where x is given locally as hat_x in K:

    T dis_evaluate (const point_basic<T>& x, size_type i_comp = 0) const;
    T operator()    (const point_basic<T>& x) const { return dis_evaluate (x,0); }
    point_basic<T> dis_vector_evaluate (const point_basic<T>& x) const;

// internals:
public:

    // evaluate uh(x) where x is given locally as hat_x in K:
    // requires to call field::dis_dof_upgrade() before.
    T evaluate (const geo_element& K, const point_basic<T>& hat_xq, size_type i_comp) const;

    // propagate changed values shared at partition boundaries to others procs
    void dis_dof_update() const;

template <class Expr>
void assembly_internal (
    const geo_basic<T,M>&      dom,
    const geo_basic<T,M>&      band,
    const band_basic<T,M>&      gh,
    const Expr&                expr,
    const quadrature_option& qopt,
    bool                       is_on_band);
template <class Expr>
void assembly (
    const geo_basic<T,M>&      domain,
    const Expr&                expr,
    const quadrature_option& qopt);

```

```

template <class Expr>
void assembly (
    const band_basic<T,M>&      gh,
    const Expr&                 expr,
    const quadrature_option& qopt);

protected:
    void dis_dof_indexes_requires_update() const;
    void dis_dof_assembly_requires_update() const;

// data:
    space_type    _V;
    mutable vec<T,M>    _u;
    mutable vec<T,M>    _b;
    mutable bool        _dis_dof_indexes_requires_update;
    mutable bool        _dis_dof_assembly_requires_update;
};
template <class T, class M>
idiststream& operator >> (odiststream& ips, field_basic<T,M>& u);

template <class T, class M>
odiststream& operator << (odiststream& ops, const field_basic<T,M>& uh);

typedef field_basic<Float> field;
typedef field_basic<Float,sequential> field_sequential;

```

4.6 field_funcutor - a functor wrapper suitable for field expressions

(Source file: 'nfem/plib/field_funcutor.h')

Description

This class is now obsolete, from Rheolef version 6.7 and is maintained for backward compatibility purpose only. Until Rheolef version 6.6, this class was used to mark functors with profil compatible with fields, i.e. that accepts **point** as parameter and returns a field value (scalar, vector, tensor). This mark was used to filter field expression arguments in **interpolate** and **integrate**. From version 6.7, this mark is no more required, and any function or functor that is callable with a **point** as argument is valid in a field expression.

A functor is a class-function, i.e. a class that defines the **operator()**. A variable **f** of a class-function can be used as **f(arg)** and when its argument is of type **point** see <undefined> [point class], page <undefined>, the function **f** interprets as a continuous field field. Thus, it can be interpolated see <undefined> [interpolate algorithm], page <undefined> and it can be combined within field expressions see <undefined> [field class], page <undefined> that appears in arguments of see <undefined> [integrate algorithm], page <undefined>.

Example

```

struct f : field_functor<f,Float> {
    Float operator() (const point& x) const { return 1-norm(x); }
};
// ...
geo omega ("square");
space Xh (omega, "P1");
field fh = interpolate (Xh, f);
test (Xh);
field lh = integrate (f*v);

```

Implementation note

The current implementation of a `field_functor` class bases on the curiously recurring template pattern (CRTP) C++ idiom: the definition of the class `f` derives from `field_functor<f,Float>` that depend itself upon `f`. So, be carrefull when using copy-paste, as there is no checks if you write e.g. `field_functor<g,Float>` with another function `g` instead of `f`.

Implementation

```

template <class Function, class Result>
struct field_functor
    : std::unary_function<point_basic<float_traits<Result> >,Result> {
    const Function& get_ref() const { return static_cast<const Function&>(*this); }
    operator Function() const { return get_ref(); }
    Result operator() (const point& x) const { return get_ref().operator()(x); }
};

```

4.7 form - representation of a finite element bilinear form

(Source file: 'nfem/plib/form.h')

Description

The `form` class groups four sparse matrix, associated to a bilinear form on two finite element spaces:

$$\begin{array}{ll}
 \mathbf{a}: U*V & \text{----> } \mathbb{R} \\
 (u,v) & \text{----> } \mathbf{a}(u,v)
 \end{array}$$

The operator \mathbf{A} associated to the bilinear form is defined by:

$$\begin{array}{ll}
 \mathbf{A}: U & \text{----> } V' \\
 u & \text{----> } \mathbf{A}(u)
 \end{array}$$

where u and v are fields (see [\[field class\]](#), page [\[undefined\]](#)), and $\mathbf{A}(u)$ is such that $\mathbf{a}(u,v) = \langle \mathbf{A}(u), v \rangle$ for all u in U and v in V and where $\langle ., . \rangle$ denotes the duality product between V and V' . Since V is a finite dimensional spaces, the duality product is the euclidian product in $\mathbb{R}^{\dim(V)}$.

Since both U and V are finite dimensional spaces, the linear operator can be represented by a matrix. The `form` class is represented by four sparse matrix in `csr` format (see [\[csr class\]](#), page [\[undefined\]](#)), associated to unknown and blocked degrees of freedom of origin and destination spaces (see [\[space class\]](#), page [\[undefined\]](#)).

Example

The operator A associated to a bilinear form $a(.,.)$ by the relation $(Au, v) = a(u, v)$ could be applied by using a sample matrix notation $A*u$, as shown by the following code:

```
geo omega("square");
space V (omega, "P1");
form a (V, V, "grad_grad");
field uh = interpolate (fct, V);
field vh = a*uh;
cout << v;
```

The form-field $vh=a*uh$ operation is equivalent to the following matrix-vector operations:

```
vh.set_u() = a.uu()*uh.u() + a.ub()*uh.b();
vh.set_b() = a.bu()*uh.u() + a.bb()*uh.b();
```

Algebra

Forms, as matrices (see [\[csr class\]](#), page [\[undefined\]](#)), support linear algebra: Adding or subtracting two forms writes $a+b$ and $a-b$, respectively, and multiplying a form by a field uh writes $a*uh$. Thus, any linear combination of forms is available.

Weighted form

A weighted form is a form with an extra weight function $w(x)$, e.g.:

$$a(uh, vh) = \int_{\Omega} \text{grad}(uh) \cdot \text{grad}(vh) w(x) dx$$

In the present implementation, w can be any field, function or class-function or any nonlinear field expression (see [\[field class\]](#), page [\[undefined\]](#)). As the integration cannot be performed exactly in general, a quadrature formula can be supplied. This feature is extensively used when solving nonlinear problems.

Implementation

```
template<class T, class M>
class form_basic {
public :
// typedefs:
```

```

typedef typename csr<T,M>::size_type    size_type;
typedef T                               value_type;
typedef typename scalar_traits<T>::type float_type;
typedef geo_basic<float_type,M>         geo_type;
typedef space_basic<float_type,M>       space_type;

// allocator/deallocator:

form_basic ();
form_basic (const form_basic<T,M>&);
form_basic<T,M>& operator= (const form_basic<T,M>&);

// allocators from initializer list (c++ 2011):

#ifdef _RHEOLEF_HAVE_STD_INITIALIZER_LIST
    form_basic (const std::initializer_list<form_concat_value<T,M> >& init_list);
    form_basic (const std::initializer_list<form_concat_line <T,M> >& init_list);
#endif // _RHEOLEF_HAVE_STD_INITIALIZER_LIST

// accessors:

const space_type& get_first_space() const;
const space_type& get_second_space() const;
const geo_type&   get_geo() const;

const communicator& comm() const;

// linear algebra:

form_basic<T,M> operator+ (const form_basic<T,M>& b) const;
form_basic<T,M> operator- (const form_basic<T,M>& b) const;
form_basic<T,M> operator* (const form_basic<T,M>& b) const;
form_basic<T,M>& operator*= (const T& lambda);
field_basic<T,M> operator* (const field_basic<T,M>& xh) const;
field_basic<T,M> trans_mult (const field_basic<T,M>& yh) const;
float_type operator () (const field_basic<T,M>& uh, const field_basic<T,M>& vh)

// io:

odiststream& put (odiststream& ops, bool show_partition = true) const;
void dump (std::string name) const;

// accessors & modifiers to unknown & blocked parts:

const csr<T,M>&    uu() const { return _uu; }
const csr<T,M>&    ub() const { return _ub; }
const csr<T,M>&    bu() const { return _bu; }
const csr<T,M>&    bb() const { return _bb; }
csr<T,M>& set_uu()      { return _uu; }

```

```

        csr<T,M>& set_ub()      { return _ub; }
        csr<T,M>& set_bu()      { return _bu; }
        csr<T,M>& set_bb()      { return _bb; }

// data
protected:
    space_type  _X;
    space_type  _Y;
    csr<T,M>    _uu;
    csr<T,M>    _ub;
    csr<T,M>    _bu;
    csr<T,M>    _bb;

// internals:
public:
    // with vf expression arg
    template <class Expr>
    void assembly_internal (
        const geo_basic<T,M>&      dom,
        const geo_basic<T,M>&      band,
        const band_basic<T,M>&     gh,
        const Expr&                expr,
        const integrate_option&    fopt,
        bool                       is_on_band);
    template <class Expr>
    void assembly (
        const geo_basic<T,M>&      domain,
        const Expr&                expr,
        const integrate_option&    fopt);
    template <class Expr>
    void assembly (
        const band_basic<T,M>&     gh,
        const Expr&                expr,
        const integrate_option&    fopt);

// backward compat: named forms
form_basic (const space_type& X, const space_type& Y,
            const std::string& name = "",
            const quadrature_option& qopt = quadrature_option());

form_basic (const space_type& X, const space_type& Y,
            const std::string& name,
            const field_basic<T,M>& weight,
            const quadrature_option& qopt = quadrature_option());

template<class Function>
form_basic (const space_type& X, const space_type& Y,
            const std::string& name,
            Function weight,

```



```

        const quadrature_option& qopt = quadrature_option());

form_basic (const space_type& X, const space_type& Y,
            const std::string& name,
            const geo_basic<T,M>& gamma,
            const quadrature_option& qopt = quadrature_option());

form_basic (const space_type& X, const space_type& Y,
            const std::string& name,
            const geo_basic<T,M>& gamma,
            const field_basic<T,M>& weight,
            const quadrature_option& qopt = quadrature_option());

template<class Function>
form_basic (
    const space_type& X,
    const space_type& Y,
    const std::string& name,
    const geo_basic<T,M>& gamma,
    Function weight,
    const quadrature_option& qopt = quadrature_option());
protected:
    // backward compat: named forms (cont.)
    template<class WeightFunction>
    void form_init (
        const std::string&      name,
        bool                    has_weight,
        WeightFunction           weight,
        const quadrature_option& qopt);
    template<class WeightFunction>
    void form_init_on_domain (
        const std::string&      name,
        const geo_basic<T,M>&    gamma,
        bool                    has_weight,
        WeightFunction           weight,
        const geo_basic<T,M>&    w_omega, // the domain where the fct wei
        const quadrature_option& qopt);
};
template<class T, class M> form_basic<T,M> trans (const form_basic<T,M>& a);
template<class T, class M> field_basic<T,M> diag (const form_basic<T,M>& a);
template<class T, class M> form_basic<T,M> diag (const field_basic<T,M>& dh);
typedef form_basic<Float,rheo_default_memory_model> form;

```

4.8 functor - a function wrapper suitable for field expressions

(Source file: 'nfem/plib/functor.h')

Description

A *functor* is a class-function, i.e. a class that defines the `operator()`: it can be used in place of an usual function. Moreover, functors can be used in Rheolef field expressions, mixed with fields (see [\[field class\]](#), page [\[undefined\]](#)). For instance, assuming that `uh` is a field and `u_ex` is a functor:

```
Float err_l1 = integrate (omega, abs(uh - uh_ex), qopt);
```

where `omega` denotes a mesh (see [\[geo class\]](#), page [\[undefined\]](#)) and `qopt` a quadrature formula (see [\[quadrature_option class\]](#), page [\[undefined\]](#)). See also the [\[integrate algorithm\]](#), page [\[undefined\]](#) function. An usual function `u_ex_f` cannot always be mixed so nicely in expressions, due to c++ language rules. For instance, the following exprtession is valid:

```
Float err_l1 = integrate (omega, abs(uh - u_ex_f), qopt);
```

In some case, the compiler cannot build a field expression using usual functionsn e.g.

```
Float I = integrate (omega, 0.5*u_ex_f), qopt);
```

because `0.5*u_ex_f` is a direct algebraic operation between usual functions and flfloating points, that is not defined in the c++ language. A way to circumvent this difficulty is to convert the usual function into a functor, as in:

```
Float I = integrate (omega, 0.5*functor(u_ex_f)), qopt);
```

Implementation

```
template<class R, class... Args>
std::function<R(Args...)>
functor (R(*f)(Args...)) {
    return std::function<R(Args...)>(f);
}
```

4.9 geo - finite element mesh

(Source file: `'nfem/plib/geo.h'`)

Synopsis

Distributed finite element mesh.

Implementation

```
template <class T>
class geo_basic<T,sequential> : public smart_pointer_clone<geo_abstract_rep<T,sequen
public:

// typedefs:
```

```

typedef sequential                                memory_type;
typedef geo_abstract_rep<T,sequential>            rep;
typedef geo_rep<T,sequential>                    rep_geo_rep;
typedef smart_pointer_clone<rep>                 base;
typedef typename rep::size_type                  size_type;
typedef typename rep::node_type                  node_type;
typedef typename rep::variant_type               variant_type;
typedef typename rep::reference                  reference;
typedef typename rep::const_reference            const_reference;
typedef typename rep::iterator                   iterator;
typedef typename rep::const_iterator             const_iterator;
typedef typename rep::iterator_by_variant        iterator_by_variant;
typedef typename rep::const_iterator_by_variant const_iterator_by_variant;
typedef typename rep::coordinate_type            coordinate_type;

// allocators:

geo_basic ();
geo_basic (std::string name, const communicator& comm = communicator());
void load (std::string name, const communicator& comm = communicator());
geo_basic (const domain_indirect_basic<sequential>& dom, const geo_basic<T,sequen

// build from_list (for level set)
geo_basic (
    const geo_basic<T,sequential>&                lambda,
    const disarray<point_basic<T>,sequential>&      node_list,
    const std::array<disarray<geo_element_auto<heap_allocator<size_type> >,sequen
        reference_element::max_variant>& elt_list)
    : base (new_macro(rep_geo_rep(lambda,node_list,elt_list))) {}

// accessors:

std::string          name() const { return base::data().name(); }
std::string          familyname() const { return base::data().familyname(); }
size_type            dimension() const { return base::data().dimension(); }
size_type            map_dimension() const { return base::data().map_dimension(); }
size_type            serial_number() const { return base::data().serial_number(); }
size_type            variant() const { return base::data().variant(); }
coordinate_type      coordinate_system() const { return base::data().coordinate_sy
std::string          coordinate_system_name() const { return space_constant::coordinate
const basis_basic<T>& get_piola_basis() const { return base::data().get_piola_b
size_type            order() const { return base::data().get_piola_bas
const node_type&      xmin() const { return base::data().xmin(); }
const node_type&      xmax() const { return base::data().xmax(); }
const T&              hmin() const { return base::data().hmin(); }
const T&              hmax() const { return base::data().hmax(); }
const distributor&    geo_element_ownership(size_type dim) const { return base::da
const geo_size&        sizes() const { return base::data().sizes(); }

```

```

const geo_size& ios_sizes() const { return base::data().ios_sizes(); }
const_reference get_geo_element (size_type dim, size_type ige) const { return base::data().get_geo_element (dim, ige); }
const_reference dis_get_geo_element (size_type dim, size_type dis_ige) const
{ return get_geo_element (dim, dis_ige); }
const geo_element& bgd2dom_geo_element (const geo_element& bgd_K) const { return base::data().bgd2dom_geo_element (bgd_K); }
const geo_element& dom2bgd_geo_element (const geo_element& dom_K) const { return base::data().dom2bgd_geo_element (dom_K); }
size_type neighbour (size_type ie, size_type loc_isid) const {
    return base::data().neighbour (ie, loc_isid); }
void neighbour_guard() const { base::data().neighbour_guard(); }
size_type n_node() const { return base::data().n_node(); }
const node_type& node(size_type inod) const { return base::data().node(inod); }
const node_type& dis_node(size_type dis_inod) const { return base::data().dis_node(dis_inod); }
void dis_inod (const geo_element& K, std::vector<size_type>& dis_inod) const {
    return base::data().dis_inod(K, dis_inod); }
node_type piola (const geo_element& K, const node_type& hat_x) const { return base::data().piola (K, hat_x); }
const disarray<node_type, sequential>& get_nodes() const { return base::data().get_nodes(); }
size_type dis_inod2dis_iv (size_type dis_inod) const { return base::data().dis_inod2dis_iv (dis_inod); }

size_type n_domain_indirect () const { return base::data().n_domain_indirect (); }
bool have_domain_indirect (const std::string& name) const { return base::data().have_domain_indirect (name); }
const domain_indirect_basic<sequential>& get_domain_indirect (size_type i) const {
    return base::data().get_domain_indirect (i); }
const domain_indirect_basic<sequential>& get_domain_indirect (const std::string& name) const {
    return base::data().get_domain_indirect (name); }
void insert_domain_indirect (const domain_indirect_basic<sequential>& dom) const {
    base::data().insert_domain_indirect (dom); }

size_type n_domain () const { return base::data().n_domain_indirect (); }
geo_basic<T, sequential> get_domain (size_type i) const;
geo_basic<T, sequential> operator[] (const std::string& name) const;
geo_basic<T, sequential> boundary() const;
geo_basic<T, sequential> internal_sides() const;
geo_basic<T, sequential> sides() const;

size_type seq_locate (
    const point_basic<T>& x,
    size_type dis_ie_guest = std::numeric_limits<size_type>::max()) const
{ return base::data().seq_locate (x, dis_ie_guest); }
size_type dis_locate (
    const point_basic<T>& x,
    size_type dis_ie_guest = std::numeric_limits<size_type>::max()) const
{ return base::data().dis_locate (x, dis_ie_guest); }
void locate (
    const disarray<point_basic<T>, sequential>& x,
    disarray<size_type, sequential>& dis_ie) const
{ return base::data().locate (x, dis_ie); }
size_type seq_trace_move (
    const point_basic<T>& x,
    const point_basic<T>& v,

```

```

        point_basic<T>&      y) const
        { return base::data().seq_trace_move (x,v,y); }

size_type dis_trace_move (
    const point_basic<T>&      x,
    const point_basic<T>&      v,
    point_basic<T>&      y) const
    { return base::data().dis_trace_move (x,v,y); }

void trace_ray_boundary (
    const disarray<point_basic<T>,sequential>&      x,
    const disarray<point_basic<T>,sequential>&      v,
    disarray<size_type, sequential>&      dis_ie,
    disarray<point_basic<T>,sequential>&      y) const
    { return base::data().trace_ray_boundary(x,v,y,dis_ie); }

void trace_move (
    const disarray<point_basic<T>,sequential>&      x,
    const disarray<point_basic<T>,sequential>&      v,
    disarray<size_type, sequential>&      dis_ie,
    disarray<point_basic<T>,sequential>&      y) const
    { return base::data().trace_move (x,v,dis_ie); }

size_type seq_nearest (
    const point_basic<T>&      x,
    point_basic<T>&      x_nearest) const
    { return base::data().seq_nearest (x, x_nearest); }

size_type dis_nearest (
    const point_basic<T>&      x,
    point_basic<T>&      x_nearest) const
    { return base::data().dis_nearest (x, x_nearest); }

void nearest (
    const disarray<point_basic<T>,sequential>&      x,
    disarray<point_basic<T>,sequential>&      x_nearest,
    disarray<size_type, sequential>&      dis_ie) const
    { base::data().nearest (x, x_nearest, dis_ie); }

// modifiers:

void set_name (std::string name);
void set_dimension (size_type dim);
void set_serial_number (size_type i);
void reset_order (size_type order);
void set_coordinate_system (coordinate_type sys_coord);
void set_coordinate_system (std::string sys_coord_name) { set_coordinate_system(sys_coord_name); }
void set_nodes (const disarray<node_type,sequential>& x);
void build_by_subdividing (const geo_basic<T,sequential>& omega, size_type k);
void build_from_data (
    const geo_header&      hdr,
    const disarray<node_type, sequential>&      node,
    std::array<disarray<geo_element_auto<heap_allocator<size_type>>,sequential>>&      tmp_geo_elems,
    bool do_upgrade);

```

```

// extended accessors:

    const communicator& comm()          const { return geo_element_ownership (0).comm; }
    size_type          size(size_type dim) const { return base::data().geo_element_ownership (dim).size; }
    size_type dis_size(size_type dim) const { return base::data().geo_element_ownership (dim).dis_size; }
    size_type          size()           const { return size      (map_dimension()); }
    size_type dis_size()           const { return dis_size (map_dimension()); }
    size_type          n_vertex()       const { return size      (0); }
    size_type dis_n_vertex()          const { return dis_size (0); }
    const_reference operator[] (size_type ie) const { return get_geo_element (map_dimension(), ie); }
    const_iterator begin (size_type dim) const { return base::data().begin(dim); }
    const_iterator end   (size_type dim) const { return base::data().end  (dim); }
    const_iterator begin ()           const { return begin(map_dimension()); }
    const_iterator end   ()           const { return end  (map_dimension()); }

    const_iterator_by_variant begin_by_variant (variant_type variant) const
    { return base::data().begin_by_variant (variant); }
    const_iterator_by_variant end_by_variant (variant_type variant) const
    { return base::data().end_by_variant (variant); }

    const geo_basic<T,sequential>& get_background_geo() const; // code in geo_domain.cpp
    geo_basic<T,sequential> get_background_domain() const;

// for compatibility with distributed interface:

    size_type ige2ios_dis_ige (size_type dim, size_type ige) const { return ige; }
    size_type dis_ige2ios_dis_ige (size_type dim, size_type dis_ige) const { return dis_ige; }
    size_type ios_ige2dis_ige (size_type dim, size_type ios_ige) const { return ios_ige; }

// comparator:

    bool operator== (const geo_basic<T,sequential>& omega2) const { return base::data().operator==(omega2); }

// i/o:

    idiststream& get (idiststream& ips);
    odiststream& put (odiststream& ops) const;
    void save (std::string filename = "") const;
    bool check (bool verbose = true) const { return base::data().check(verbose); }
};

```

Implementation

```

template <class T>
class geo_basic<T,distributed> : public smart_pointer_clone<geo_abstract_rep<T,distributed>>
public:

// typedefs:

```

```

typedef distributed                memory_type;
typedef geo_abstract_rep<T,distributed> rep;
typedef geo_rep<T,distributed>     rep_geo_rep;
typedef smart_pointer_clone<rep>   base;
typedef typename rep::size_type    size_type;
typedef typename rep::node_type    node_type;
typedef typename rep::variant_type variant_type;
typedef typename rep::node_map_type node_map_type;
typedef typename rep::reference     reference;
typedef typename rep::const_reference const_reference;
typedef typename rep::iterator      iterator;
typedef typename rep::const_iterator const_iterator;
typedef typename rep::iterator_by_variant iterator_by_variant;
typedef typename rep::const_iterator_by_variant const_iterator_by_variant;
typedef typename rep::coordinate_type coordinate_type;

// allocators:

geo_basic ();
geo_basic (std::string name, const communicator& comm = communicator());
void load (std::string name, const communicator& comm = communicator());
geo_basic (const domain_indirect_basic<distributed>& dom, const geo_basic<T,distributed>& dom_geo) {
    // build from_list (for level set)
    geo_basic (
        const geo_basic<T,distributed>& lambda,
        const disarray<point_basic<T>,distributed>& node_list,
        const std::array<disarray<geo_element_auto<heap_allocator<size_type> >,distributed>& elt_list,
            reference_element::max_variant>& elt_list)
        : base (new_macro(rep_geo_rep(lambda,node_list,elt_list))) {}

// accessors:

std::string name() const { return base::data().name(); }
std::string familyname() const { return base::data().familyname(); }
size_type dimension() const { return base::data().dimension(); }
size_type map_dimension() const { return base::data().map_dimension(); }
size_type serial_number() const { return base::data().serial_number(); }
size_type variant() const { return base::data().variant(); }
coordinate_type coordinate_system() const { return base::data().coordinate_system(); }
std::string coordinate_system_name() const { return space_constant::coordinate_system_name(); }
const basis_basic<T>& get_piola_basis() const { return base::data().get_piola_basis(); }
size_type order() const { return base::data().get_piola_basis().order(); }
const node_type& xmin() const { return base::data().xmin(); }
const node_type& xmax() const { return base::data().xmax(); }
const T& hmin() const { return base::data().hmin(); }
const T& hmax() const { return base::data().hmax(); }
const distributor& geo_element_ownership(size_type dim) const

```

```

        { return base::data().geo_element_ownership (dim); }
const geo_size&      sizes()                const { return base::data().sizes(); }
const geo_size&  ios_sizes()                const { return base::data().ios_sizes(); }
const_reference get_geo_element (size_type dim, size_type ige) const
    { return base::data().get_geo_element (dim, ige); }
const_reference dis_get_geo_element (size_type dim, size_type dis_ige) const
    { return base::data().dis_get_geo_element (dim, dis_ige); }
const geo_element& bgd2dom_geo_element (const geo_element& bgd_K) const
    { return base::data().bgd2dom_geo_element (bgd_K); }
const geo_element& dom2bgd_geo_element (const geo_element& dom_K) const
    { return base::data().dom2bgd_geo_element (dom_K); }
size_type neighbour (size_type ie, size_type loc_isid) const {
    return base::data().neighbour (ie, loc_isid); }
void neighbour_guard() const { base::data().neighbour_guard(); }
distributor geo_element_ios_ownership (size_type dim) const {
    return base::data().geo_element_ios_ownership (dim); }
size_type ige2ios_dis_ige (size_type dim, size_type ige) const {
    return base::data().ige2ios_dis_ige (dim, ige); }
size_type dis_ige2ios_dis_ige (size_type dim, size_type dis_ige) const {
    return base::data().dis_ige2ios_dis_ige (dim, dis_ige); }
size_type ios_ige2dis_ige (size_type dim, size_type ios_ige) const {
    return base::data().ios_ige2dis_ige (dim, ios_ige); }
size_type      n_node() const { return base::data().n_node(); }
const node_type&      node(size_type      inod) const { return base::data().node(inod); }
const node_type& dis_node(size_type dis_inod) const { return base::data().dis_node(dis_inod); }
void dis_inod (const geo_element& K, std::vector<size_type>& dis_inod) const {
    return base::data().dis_inod(K, dis_inod); }
node_type piola (const geo_element& K, const node_type& hat_x) const { return base::data().piola(K, hat_x); }
const disarray<node_type, distributed>& get_nodes() const { return base::data().get_nodes(); }

size_type n_domain_indirect () const { return base::data().n_domain_indirect (); }
bool have_domain_indirect (const std::string& name) const { return base::data().have_domain_indirect (name); }
const domain_indirect_basic<distributed>& get_domain_indirect (size_type i) const {
    return base::data().get_domain_indirect (i); }
const domain_indirect_basic<distributed>& get_domain_indirect (const std::string& name) const {
    return base::data().get_domain_indirect (name); }
void insert_domain_indirect (const domain_indirect_basic<distributed>& dom) const {
    base::data().insert_domain_indirect (dom); }

size_type n_domain () const { return base::data().n_domain_indirect (); }
geo_basic<T, distributed> get_domain (size_type i) const;
geo_basic<T, distributed> operator[] (const std::string& name) const;
geo_basic<T, distributed> boundary() const;
geo_basic<T, distributed> internal_sides() const;
geo_basic<T, distributed> sides() const;

size_type seq_locate (
    const point_basic<T>& x,
    size_type dis_ie_guest = std::numeric_limits<size_type>::max()) const;

```



```

        { return base::data().seq_locate (x, dis_ie_guest); }
size_type dis_locate (
    const point_basic<T>& x,
    size_type dis_ie_guest = std::numeric_limits<size_type>::max()) const
{ return base::data().dis_locate (x, dis_ie_guest); }
void locate (const disarray<point_basic<T>, distributed>& x, disarray<size_type>
{ return base::data().locate (x, dis_ie); }
size_type seq_trace_move (
    const point_basic<T>&      x,
    const point_basic<T>&      v,
    point_basic<T>&            y) const
{ return base::data().seq_trace_move (x,v,y)
size_type dis_trace_move (
    const point_basic<T>&      x,
    const point_basic<T>&      v,
    point_basic<T>&            y) const
{ return base::data().dis_trace_move (x,v,y)
void trace_ray_boundary (
    const disarray<point_basic<T>,distributed>&      x,
    const disarray<point_basic<T>,distributed>&      v,
    disarray<size_type, distributed>&                  dis_ie,
    disarray<point_basic<T>,distributed>&              y) const
{ return base::data().trace_ray_boundary (x,v,y)
void trace_move (
    const disarray<point_basic<T>,distributed>&      x,
    const disarray<point_basic<T>,distributed>&      v,
    disarray<size_type, distributed>&                  dis_ie,
    disarray<point_basic<T>,distributed>&              y) const
{ return base::data().trace_move (x,v,dis_ie)
size_type seq_nearest (
    const point_basic<T>&      x,
    point_basic<T>&            x_nearest) const
{ return base::data().seq_nearest (x, x_nearest)
size_type dis_nearest (
    const point_basic<T>&      x,
    point_basic<T>&            x_nearest) const
{ return base::data().dis_nearest (x, x_nearest)
void nearest (
    const disarray<point_basic<T>,distributed>&      x,
    disarray<point_basic<T>,distributed>&              x_nearest,
    disarray<size_type, distributed>&                  dis_ie) const
{ base::data().nearest (x, x_nearest, dis_ie)
// modifiers:

void set_nodes (const disarray<node_type,distributed>& x);
void reset_order (size_type order);
size_type dis_inod2dis_iv (size_type dis_inod) const { return base::data().dis_inod2dis_iv (dis_inod);
void set_coordinate_system (coordinate_type sys_coord);
void set_coordinate_system (std::string sys_coord_name) { set_coordinate_system (sys_coord_name);

```

```

void set_dimension (size_type dim);
void set_serial_number (size_type i);
void set_name (std::string name);
void build_by_subdividing (const geo_basic<T,distributed>& omega, size_type k);

// extended accessors:

size_type      size(size_type dim) const { return base::data().geo_element_owner;
size_type dis_size(size_type dim) const { return base::data().geo_element_owner;
const communicator& comm()          const { return geo_element_ownership (0).comm
size_type      size()                const { return size      (map_dimension()); }
size_type dis_size()                const { return dis_size (map_dimension()); }
size_type      n_vertex()            const { return size      (0); }
size_type dis_n_vertex()            const { return dis_size (0); }
const_reference operator[] (size_type ie) const
    { return get_geo_element (map_dimension(), ie); }

const_iterator begin (size_type dim) const { return base::data().begin(dim); }
const_iterator end   (size_type dim) const { return base::data().end  (dim); }
const_iterator begin ()          const { return begin(map_dimension()); }
const_iterator end   ()          const { return end  (map_dimension()); }

const_iterator_by_variant begin_by_variant (variant_type variant) const
    { return base::data().begin_by_variant (variant); }
const_iterator_by_variant end_by_variant (variant_type variant) const
    { return base::data().end_by_variant (variant); }

const geo_basic<T,distributed>& get_background_geo() const; // code in geo_domai
geo_basic<T,distributed> get_background_domain() const;

// comparator:

bool operator== (const geo_basic<T,distributed>& omega2) const { return base::d

// i/o:

odiststream& put (odiststream& ops) const { return base::data().put (ops); }
idiststream& get (idiststream& ips);
void save (std::string filename = "") const;
bool check (bool verbose = true) const { return base::data().check(verbose); }

// utilities:

void set_ios_permutation (
    std::array<size_type,reference_element::max_variant>& loc_ndof_by_variant,
    disarray<size_type,distributed>& idof2ios_dis_idof) co
    { base::data().set_ios_permutation (loc_ndof_by_variant, idof2ios_dis_idof); }
};

```

4.10 integrate_option - send options to the integrate function

(Source file: 'nfem/plib/integrate_option.h')

Description

This class is used to send options to the integrate function when building a form. It allows one to set the quadrature formulae that is used for numerical integration see [\[quadrature_option class\]](#), page [\[undefined\]](#) and two boolean flags.

Flags

ignore_sys_coord

This flag has effects only for axisymmetric coordinate systems. When set, it omits the r weight in the $r \, dr \, dz$ measure during the numerical integration performed the `integrate` function. This feature is useful for computing the stream function in the axisymmetric case.

lump This flag, when set, performs a *mass lumping procedure* on the matrix at the element level:

$$a(i,i) := \sum(j) a(i,j)$$

The resulting matrix is diagonal. This feature is useful for computing a diagonal approximation of the mass matrix for the continuous P1 element.

invert This flag, when set, performs a *local inversion* on the matrix at the element level:

$$a := \text{inv}(a)$$

This procedure is allowed only when the global matrix is block diagonal, e.g. for discontinuous or bubble approximations. This property is true when basis functions have a compact support inside exactly one element.

Default flag values

All flags are set to false by default.

Implementation

```
struct integrate_option : quadrature_option {
// allocators:
  integrate_option();
  integrate_option (const integrate_option& fopt);
  integrate_option (const quadrature_option& qopt);
  integrate_option& operator= (const integrate_option& fopt);
// data:
  bool ignore_sys_coord, lump, invert;
};
```

4.11 space – piecewise polynomial finite element space

(Source file: ‘nfem/plib/space.h’)

Description

The `space` class contains some numbering for unknowns and blocked degrees of freedoms related to a given mesh and polynomial approximation.

Synopsis

```
space Q (omega, "P1");
space V (omega, "P2", "vector");
space T (omega, "P1d", "tensor");
```

Product

```
space X = T*V*Q;
space Q2 = pow(Q,2);
```

Implementation

```
template <class T>
class space_basic<T,sequential> : public smart_pointer<space_rep<T,sequential>> > {
public:

// typedefs:

    typedef space_rep<T,sequential>    rep;
    typedef smart_pointer<rep>          base;
    typedef typename rep::size_type    size_type;
    typedef typename rep::valued_type  valued_type;

// allocators:

    space_basic (const geo_basic<T,sequential>& omega = (geo_basic<T,sequential>())
                std::string approx          = "",
                std::string valued          = "scalar");
    space_basic (const space_mult_list<T,sequential>& expr);
    space_basic (const space_constitution<T,sequential>& constit);

// accessors:

    void block (std::string dom_name);
    void unblock(std::string dom_name);
    void block (const domain_indirect_basic<sequential>& dom);
    void unblock(const domain_indirect_basic<sequential>& dom);

    const distributor& ownership() const;
    const communicator& comm() const;
```

```

size_type          ndof() const;
size_type          dis_ndof() const;

const geo_basic<T,sequential>& get_geo() const;
const numbering<T,sequential>& get_numbering() const;
size_type size() const;
valued_type        valued_tag()    const;
const std::string&  valued()        const;
space_component<T,sequential>      operator[] (size_type i_comp);
space_component_const<T,sequential> operator[] (size_type i_comp) const;
const space_constitution<T,sequential>& get_constitution() const;
size_type degree() const;
std::string get_approx() const;
std::string stamp() const;

void dis_idof (const geo_element& K, std::vector<size_type>& dis_idof) const;

const distributor& iu_ownership() const;
const distributor& ib_ownership() const;

bool      is_blocked (size_type      idof) const;
size_type iub (size_type      idof) const;
bool dis_is_blocked (size_type dis_idof) const;
size_type dis_iub (size_type dis_idof) const;

const distributor& ios_ownership() const;
size_type idof2ios_dis_idof (size_type idof) const;
size_type ios_idof2dis_idof (size_type ios_idof) const;

const point_basic<T>& xdof (size_type idof) const;
const disarray<point_basic<T>,sequential>& get_xdofs() const;

template <class Function>
T momentum (const Function& f, size_type idof) const;

template <class Function>
point_basic<T> vector_momentum (const Function& f, size_type idof) const;

template <class Function>
tensor_basic<T> tensor_momentum (const Function& f, size_type idof) const;

disarray<size_type, sequential> build_indirect_array (
    const space_basic<T,sequential>& Wh, const std::string& dom_name) const;

disarray<size_type, sequential> build_indirect_array (
    const space_basic<T,sequential>& Wh, const geo_basic<T,sequential>& bgd_gam

const std::set<size_type>& ext_iu_set() const { return base::data().ext_iu_set(
const std::set<size_type>& ext_ib_set() const { return base::data().ext_ib_set(

```

```

// comparator:

    bool operator== (const space_basic<T,sequential>& V2) const { return base::data
    bool operator!= (const space_basic<T,sequential>& V2) const { return ! operator
    friend bool are_compatible (const space_basic<T,sequential>& V1, const space_ba
        return are_compatible (V1.data(), V2.data()); }
};

```

Implementation

```

template <class T>
class space_basic<T,distributed> : public smart_pointer<space_rep<T,distributed> >
public:

// typedefs:

    typedef space_rep<T,distributed>    rep;
    typedef smart_pointer<rep>          base;
    typedef typename rep::size_type     size_type;
    typedef typename rep::valued_type   valued_type;

// allocators:

    space_basic (const geo_basic<T,distributed>& omega = (geo_basic<T,distributed>()
        std::string approx          = "",
        std::string valued          = "scalar");
    space_basic (const space_mult_list<T,distributed>&);
    space_basic (const space_constitution<T,distributed>& constit);

// accessors:

    void block (std::string dom_name);
    void unblock(std::string dom_name);
    void block (const domain_indirect_basic<distributed>& dom);
    void unblock(const domain_indirect_basic<distributed>& dom);

    const distributor& ownership() const;
    const communicator& comm() const;
    size_type         ndof() const;
    size_type         dis_ndof() const;

    const geo_basic<T,distributed>& get_geo() const;
    const numbering<T,distributed>& get_numbering() const;
    size_type size() const;
    valued_type         valued_tag() const;
    const std::string&   valued() const;
    space_component<T,distributed> operator[] (size_type i_comp);
    space_component_const<T,distributed> operator[] (size_type i_comp) const;

```

```

const space_constitution<T,distributed>& get_constitution() const;
size_type degree() const;
std::string get_approx() const;
std::string stamp() const;

void dis_idof (const geo_element& K, std::vector<size_type>& dis_idof) const;

const distributor& iu_ownership() const;
const distributor& ib_ownership() const;

bool      is_blocked (size_type      idof) const;
size_type      iub (size_type      idof) const;

bool  dis_is_blocked (size_type dis_idof) const;
size_type      dis_iub (size_type dis_idof) const;

const distributor& ios_ownership() const;
size_type idof2ios_dis_idof (size_type idof) const;
size_type ios_idof2dis_idof (size_type ios_idof) const;

const point_basic<T>& x dof (size_type idof) const;
const disarray<point_basic<T>,distributed>& get_xdofs() const;

template <class Function>
T momentum (const Function& f, size_type idof) const;

template <class Function>
point_basic<T> vector_momentum (const Function& f, size_type idof) const;

template <class Function>
tensor_basic<T> tensor_momentum (const Function& f, size_type idof) const;

disarray<size_type, distributed> build_indirect_array (
    const space_basic<T,distributed>& Wh, const std::string& dom_name) const;

disarray<size_type, distributed> build_indirect_array (
    const space_basic<T,distributed>& Wh, const geo_basic<T,distributed>& bgd_g

const std::set<size_type>& ext_iu_set() const { return base::data().ext_iu_set(
const std::set<size_type>& ext_ib_set() const { return base::data().ext_ib_set(

// comparator:

bool operator== (const space_basic<T,distributed>& V2) const { return base::dat
bool operator!= (const space_basic<T,distributed>& V2) const { return ! operator
friend bool are_compatible (const space_basic<T,distributed>& V1, const space_b
    return are_compatible (V1.data(), V2.data()); }
};

```

4.12 test, trial - symbolic arguments in variational expressions

(Source file: 'nfem/plib/test.h')

Description

These classes are used for test and trial functions involved in variational formulations. Variational formulations could be specified by expressions, in the spirit of c++ embedded languages. A variable of the `test` type represents the formal argument (the test-function) in the definition of a linear form, as in:

```
geo omega ("circle");
space Xh (omega, "P1");
test v(Xh);
field lh = integrate (omega, 2*v);
```

For a bilinear form, the test-function represents its second formal argument, while the first one is designed by the `trial` type:

```
trial u(Xh);
test v(Xh);
form m = integrate (omega, u*v),
      a = integrate (omega, dot(grad(u),grad(v)));
```

The `field_vf` class is able to represent either `test` or `trial` types: it could be used to formally define a differential operator, as:

```
D(u) = (grad(u)+trans(grad(u)))
```

4.13 ad3 - automatic differentiation with respect to variables in R^3

(Source file: 'nfem/gbasis/ad3.h')

Description

The `ad3` class defines a forward automatic differentiation in R^3 for computing automatically the gradient of a function from R^3 to R . The implementation uses a simple forward automatic differentiation method.

Example

```
template<class T>
T f (const point_basic<T>& x) { return sqr(x[0]) + x[0]*x[1] + 1/x[2]; }
...
point_basic<ad3> x_ad = ad3::point (x);
ad3 y_ad = f(x_ad);
cout << "f          ="<<y_ad <<endl
      << "gradq(f)="<<y_ad.grad() <<endl;
```


4.14 basis - polynomial basis

(Source file: 'nfem/gbasis/basis.h')

Synopsis

The **basis** class defines functions that evaluates a polynomial basis and its derivatives on a point. The polynomial basis is designated by a string, e.g. "P0", "P1", "P2", "bubble",... indicating the basis family and the polynomial degree. The basis depends also of the reference element: triangle, square, tetrahedron, etc (see [\[reference_element iclass\]](#), page [\[undefined\]](#)). For instance, on the square reference element, the "P1" string designates the common Q1 four-nodes basis on the reference square. The basis is evaluated by using the **eval** member function. The nodes associated to the Lagrange polynomial basis are also available by the member function **hat_node**. The Lagrange basis family is designated by **Pk**, where **k** is any polynomial order.

In addition to the usual nodal Lagrange family, two non-nodal basis are provided: the Bernstein family, designated by **Bk** and the spectral Dubiner family, designated by **Sk** (see Dubiner, 1991 J. Sci. Comput.). By contrast, these two non-nodal basis are called modal. For these modal basis, the nodes provided by the member function **hat_node** represent the nodes used by the interpolation operator onto this basis.

The vector-valued Raviart-Thomas family is designated by **RTk**.

Options

The Lagrange basis recognize some options, transmitted to the constructor of the basis class: see [\[basis_option class\]](#), page [\[undefined\]](#).

Notes

The **basis** class is a [\[smart_pointer iclass\]](#), page [\[undefined\]](#) class on a **basis_rep** class that is a pure virtual base class for effective bases, e.g. **Pk_lagrange**, **Pk_bernstein**, **RTk**, etc.

Limitations

The **Sk** and **RTk** basis are still under development.

Implementation

```
template<class T>
class basis_basic : public smart_pointer_nocopy<basis_rep<T> > {
public:

// typedefs:

    typedef basis_rep<T>          rep;
    typedef smart_pointer_nocopy<rep> base;
    typedef typename rep::size_type size_type;
```

```

typedef typename rep::valued_type valued_type;

// allocators:

basis_basic (std::string name = "");
void reset (std::string& name);

// accessors:

std::string family_name() const;
size_type degree() const;
std::string name() const;
size_type size (reference_element hat_K) const;
bool is_continuous() const;
bool is_discontinuous() const;
const basis_option& option() const;
valued_type valued_tag() const;
const std::string& valued() const;

void get_local_idof_on_side (
    reference_element hat_K,
    const side_information_type& sid,
    arma::Col<size_type>& loc_idof) const;

// evaluate the basis:

void eval(
    reference_element hat_K,
    const point_basic<T>& hat_x,
    arma::Col<T>& value) const; // scalar-valued
void eval(
    reference_element hat_K,
    const point_basic<T>& hat_x,
    std::vector<point_basic<T> >& value) const; // vector-valued

// evaluate it on a node set:
void eval(
    reference_element hat_K,
    const std::vector<point_basic<T> >& nodes,
    arma::Mat<T>& values) const; // scalar-valued
void eval(
    reference_element hat_K,
    const std::vector<point_basic<T> >& nodes,
    ublas::matrix<point_basic<T> >& values) const; // vector-valued

// same, but matrix flattened in a linear arrow of size nrow*ncol
void eval(
    reference_element hat_K,
    const std::vector<point_basic<T> >& nodes,

```

```

        arma::Col<T>&                                values) const; // scalar-valued

// evaluate the gradient of the basis:

void grad_eval(
    reference_element          hat_K,
    const point_basic<T>&      hat_x,
    std::vector<point_basic<T> >& value) const; // scalar-valued
void grad_eval(
    reference_element          hat_K,
    const point_basic<T>&      hat_x,
    std::vector<tensor_basic<T> >& value) const; // vector-valued

// evaluate it on a node set:
void grad_eval(
    reference_element          hat_K,
    const std::vector<point_basic<T> >& nodes,
    ublas::matrix<point_basic<T> >& values) const; // scalar-valued
void grad_eval(
    reference_element          hat_K,
    const std::vector<point_basic<T> >& nodes,
    ublas::matrix<tensor_basic<T> >& values) const; // vector-valued

// same, but matrix flatened in a linear arrow of size nrow*ncol
void grad_eval(
    reference_element          hat_K,
    const std::vector<point_basic<T> >& nodes,
    std::vector<point_basic<T> >& values) const; // scalar-valued

// compute_dof:

template <class Function>
void compute_dof (
    reference_element          hat_K,
    const Function&            f,
    arma::Col<T>&              dofs) const;

const arma::Mat<T>& vdm      (reference_element hat_K) const;
const arma::Mat<T>& inv_vdm (reference_element hat_K) const;

// output:

void put          (std::ostream& os, reference_element hat_K) const;
void put_hat_node (std::ostream& os, reference_element hat_K) const;
void put_hat_node_on_side (
    std::ostream&          os,
    reference_element      hat_K,
    const side_information_type& sid) const;

```

```

        const std::vector<point_basic<T> >& hat_node (reference_element hat_K) const;

protected:
// internals:

        void _clear() const;
};
typedef basis_basic<Float> basis;

```

4.15 basis_option - options of the finite element space

(Source file: 'nfem/gbasis/basis_option.h')

Description

The **basis_option** class is used to set options for the **space** constructor of the finite element space (see [\[space class\]](#), page [\[undefined\]](#)). These options are directly transmitted to the **basis** class when computing the polynomial basis (see [\[basis class\]](#), page [\[undefined\]](#) and [\[basis command\]](#), page [\[undefined\]](#)). There are two main options: Lagrange nodes and raw polynomial basis.

Lagrange nodes

There are two possible node sets. The **equispaced**, refers to an equispaced set of Lagrange node during interpolation. The **warburton** one, refers to a non-equispaced optimized set of Lagrange nodes, suitable for high order polynomials (see Warburton, 2006, J. Eng. Math.). With this choice, the interpolation error is dramatically decreased for high order polynomials. The default is the **equispaced** node set, for backward compatibility purpose.

Continuous feature

Some elements are possibly continuous. For instance, **Pk** elements family, with $k \geq 1$, possibly have continuous interelements reconnections and thus, could define a piecewise polynomial and globally continuous function. These basis families can be used both as continuous or discontinuous elements. Nevertheless, this choice should be done when building the basis: the information is transmitted during basis construction by sending the **basis_option** class and setting appropriately this feature with the **set_continuous(c)** member function.

Raw polynomial basis

The raw (or initial) basis is used for building the Lagrange basis, via the Vandermonde matrix and its inverse. Its choice do not change the definition of the FEM basis, but only the way it is builded. There are three possible raw basis. The **monomial** basis is the simplest choice, suitable for low order polynomials (less than five). For higher order polynomial, the Vandermonde matrix becomes ill-conditioned and its inverse leads to errors in double precision. The **dubiner** basis (see Dubiner, 1991 J. Sci. Comput.) leads to better condition number. The **bernstein** basis could also be used as an alternative raw basis. The default is the **dubiner** raw basis.

String option representation

The `is_option(string)` and `set(string)` members leads to easy setting of combined options at run time. By this way, options can be specified, together with basis basename, on the command line or from a file.

The `stamp()` member returns a unique string. This string is used for specifying basis options, e.g. on command line or in files. This string is empty when all options are set to default values. Otherwise, it returns a comma separated list of options, enclosed by braces, specifying only non-default options. For instance, combining Warburton node set and Dubiner raw polynomials leads to "`@{warburton@}`". Also, combining Warburton node set and Bernstein raw polynomials leads to "`@{warburton,bernstein@}`".

Note that the continuous or discontinuous feature is not specified by the `stamp()` string: it will be specified into the basis basename, by appending a "d" letter, as in "`P6d@{warburton@}`".

Notes

There are two distinct kind of polynomial basis: the raw basis and the finite element one. (see [\[basis class\]](#), page [\[undefined\]](#) and [\[basis command\]](#), page [\[undefined\]](#)). When using the Pk Lagrange finite element basis, these options are used to transform from one raw (initial) polynomial basis to the Lagrange one, based on a node set. When using an alternative finite element basis, e.g. the spectral Sk or the Bernstein Bk, these options do not have any effect.

Implementation

```
class basis_option {
public:
// typedefs:

    typedef size_t size_type;

    typedef enum {
        equispaced = 0,
        warburton   = 1,
        fekete      = 2,
        max_node    = 3
    } node_type; // update also node_name[]

    typedef enum {
        monomial          = 0,
        bernstein         = 1,
        dubiner           = 2,
        max_raw_polynomial = 3
    } raw_polynomial_type; // update also raw_polynomial_name[]

    static const node_type      default_node      = basis_option::equispaced;
    static const raw_polynomial_type default_raw_polynomial = basis_option::dubiner;
```

```

// allocators:

basis_option(
    node_type          nt = default_node,
    raw_polynomial_type pt = default_raw_polynomial);

basis_option (const basis_option& sopt);
basis_option& operator= (const basis_option& sopt);

// accessors & modifiers:

node_type          get_node() const;
raw_polynomial_type get_raw_polynomial() const;
std::string        get_node_name() const;
std::string        get_raw_polynomial_name() const;
bool               is_continuous() const;
bool               is_discontinuous() const;

void set_node      (node_type type);
void set_raw_polynomial (raw_polynomial_type type);
void set           (std::string option_name);
void set_node      (std::string node_name);
void set_raw_polynomial (std::string raw_polynomial_name);
void set_continuous (bool c = true);
void set_discontinuous (bool c = false);

bool is_node_name      (std::string name) const;
bool is_raw_polynomial_name (std::string name) const;
bool is_option_name     (std::string name) const;

std::string stamp() const;

// data:
protected:
    node_type          _node;
    raw_polynomial_type _poly;
    bool               _is_continuous;
};

```

4.16 point - vertex of a mesh

(Source file: 'nfem/geo_element/point.h')

Description

Defines geometrical vertex as an array of coordinates. This array is also used as a vector of the three dimensional physical space.

Implementation

```

template <class T>
class point_basic {
    public:

// typedefs:

        typedef size_t size_type;
        typedef T      element_type;
        typedef T      scalar_type;
        typedef T      float_type;

// allocators:

        explicit point_basic () { _x[0] = T(); _x[1] = T(); _x[2] = T(); }

        explicit point_basic (
            const T& x0,
            const T& x1 = 0,
            const T& x2 = 0)
            { _x[0] = x0; _x[1] = x1; _x[2] = x2; }

        template <class T1>
        point_basic<T>(const point_basic<T1>& p)
            { _x[0] = p._x[0]; _x[1] = p._x[1]; _x[2] = p._x[2]; }

        template <class T1>
        point_basic<T>& operator = (const point_basic<T1>& p)
            { _x[0] = p._x[0]; _x[1] = p._x[1]; _x[2] = p._x[2]; return *this; }

#ifdef _RHEOLEF_HAVE_STD_INITIALIZER_LIST
        point_basic (const std::initializer_list<T>& il);
#endif // _RHEOLEF_HAVE_STD_INITIALIZER_LIST

// accessors:

        T& operator[] (int i_coord) { return _x[i_coord%3]; }
        const T& operator[] (int i_coord) const { return _x[i_coord%3]; }
        T& operator() (int i_coord) { return _x[i_coord%3]; }
        const T& operator() (int i_coord) const { return _x[i_coord%3]; }

        // interface for CGAL library inter-operability:
        const T& x() const { return _x[0]; }
        const T& y() const { return _x[1]; }
        const T& z() const { return _x[2]; }
        T& x(){ return _x[0]; }
        T& y(){ return _x[1]; }
        T& z(){ return _x[2]; }

// inputs/outputs:

```

```

std::istream& get (std::istream& s, int d = 3)
{
    switch (d) {
        case 0 : _x[0] = _x[1] = _x[2] = T(0); return s;
        case 1 : _x[1] = _x[2] = T(0); return s >> _x[0];
        case 2 : _x[2] = T(0); return s >> _x[0] >> _x[1];
        default: return s >> _x[0] >> _x[1] >> _x[2];
    }
}
// output
std::ostream& put (std::ostream& s, int d = 3) const;

// algebra:

bool operator== (const point_basic<T>& v) const
    { return _x[0] == v[0] && _x[1] == v[1] && _x[2] == v[2]; }

bool operator!= (const point_basic<T>& v) const
    { return !operator==(v); }

point_basic<T>& operator+= (const point_basic<T>& v)
    { _x[0] += v[0]; _x[1] += v[1]; _x[2] += v[2]; return *this; }

point_basic<T>& operator-= (const point_basic<T>& v)
    { _x[0] -= v[0]; _x[1] -= v[1]; _x[2] -= v[2]; return *this; }

point_basic<T>& operator*= (const T& a)
    { _x[0] *= a; _x[1] *= a; _x[2] *= a; return *this; }

point_basic<T>& operator/= (const T& a)
    { _x[0] /= a; _x[1] /= a; _x[2] /= a; return *this; }

point_basic<T> operator+ (const point_basic<T>& v) const
    { return point_basic<T> (_x[0]+v[0], _x[1]+v[1], _x[2]+v[2]); }

point_basic<T> operator- () const
    { return point_basic<T> (-_x[0], -_x[1], -_x[2]); }

point_basic<T> operator- (const point_basic<T>& v) const
    { return point_basic<T> (_x[0]-v[0], _x[1]-v[1], _x[2]-v[2]); }

template <class U>
typename
std::enable_if<
    details::is_rheolef_arithmetic<U>::value
,point_basic<T>
>::type
operator* (const U& a) const

```



```

        { return point_basic<T> (_x[0]*a, _x[1]*a, _x[2]*a); }
point_basic<T> operator/ (const T& a) const
{ return operator* (T(1)/T(a)); }
point_basic<T> operator/ (point_basic<T> v) const
{ return point_basic<T> (_x[0]/v[0], _x[1]/v[1], _x[2]/v[2]); }

// data:
// protected:
    T _x[3];
// internal:
    static T _my_abs(const T& x) { return (x > T(0)) ? x : -x; }
};
typedef point_basic<Float> point;

// algebra:
template <class T, class U>
inline
typename
std::enable_if<
    details::is_rheolef_arithmetic<U>::value
    ,point_basic<T>
>::type
operator* (const U& a, const point_basic<T>& u)
{
    return point_basic<T> (a*u[0], a*u[1], a*u[2]);
}
template<class T>
inline
point_basic<T>
vect (const point_basic<T>& v, const point_basic<T>& w)
{
    return point_basic<T> (
        v[1]*w[2]-v[2]*w[1],
        v[2]*w[0]-v[0]*w[2],
        v[0]*w[1]-v[1]*w[0]);
}
// metrics:
template<class T>
inline
T dot (const point_basic<T>& x, const point_basic<T>& y)
{
    return x[0]*y[0]+x[1]*y[1]+x[2]*y[2];
}
template<class T>
inline
T norm2 (const point_basic<T>& x)
{
    return dot(x,x);
}

```

```

template<class T>
inline
T norm (const point_basic<T>& x)
{
    return sqrt(norm2(x));
}
template<class T>
inline
T dist2 (const point_basic<T>& x,  const point_basic<T>& y)
{
    return norm2(x-y);
}
template<class T>
inline
T dist (const point_basic<T>& x,  const point_basic<T>& y)
{
    return norm(x-y);
}
template<class T>
inline
T dist_infty (const point_basic<T>& x,  const point_basic<T>& y)
{
    return max(point_basic<T>::_my_abs(x[0]-y[0]),
               max(point_basic<T>::_my_abs(x[1]-y[1]),
                   point_basic<T>::_my_abs(x[2]-y[2])));
}
template <class T>
T vect2d (const point_basic<T>& v, const point_basic<T>& w);

template <class T>
T mixt (const point_basic<T>& u, const point_basic<T>& v, const point_basic<T>& w);

// robust(exact) floating point predicates: return the sign of the value as (0, > 0
// formally: orient2d(a,b,x) = vect2d(a-x,b-x)
template <class T>
int
sign_orient2d (
    const point_basic<T>& a,
    const point_basic<T>& b,
    const point_basic<T>& c);

template <class T>
int
sign_orient3d (
    const point_basic<T>& a,
    const point_basic<T>& b,
    const point_basic<T>& c,
    const point_basic<T>& d);

```

```

// compute also the value:
template <class T>
T orient2d(
    const point_basic<T>& a,
    const point_basic<T>& b,
    const point_basic<T>& c);

// formally: orient3d(a,b,c,x) = mixt3d(a-x,b-x,c-x)
template <class T>
T orient3d(
    const point_basic<T>& a,
    const point_basic<T>& b,
    const point_basic<T>& c,
    const point_basic<T>& d);

template <class T>
std::string ptos (const point_basic<T>& x, int d = 3);

// ccomparators: lexicographic order
template<class T, size_t d>
bool
lexicographically_less (const point_basic<T>& a, const point_basic<T>& b)
{
    for (typename point_basic<T>::size_type i = 0; i < d; i++) {
        if (a[i] < b[i]) return true;
        if (a[i] > b[i]) return false;
    }
    return false; // equality
}

```

4.17 quadrature_option - send options to the integrate function

(Source file: 'nfem/geo_element/quadrature_option.h')

Synopsis

The `quadrature_option` class is used to send options to the `integrate` function see [\(undefined\)](#) [integrate algorithm], page [\(undefined\)](#). This class allows one to set the family (Gauss, Gauss-Lobatto, etc) and the polynomial degree that is exactly integrated.

A special family name, call `equispaced`, refers to an equispaced set of point, suitable for some irregular functions, e.g. the Heaviside function. In that case, the `order` parameter refers to the number of nodes used. For instance, `order=1` refers to the trapezoidal formulae and for the general case, there are `order+1` nodes per edges. See also the see [\(undefined\)](#) [quadrature iclass], page [\(undefined\)](#) for more on quadrature formulae.

Implementation

```

class quadrature_option {
public:
// typedefs:

    typedef size_t size_type;

    typedef enum {
        gauss            = 0,
        gauss_lobatto    = 1,
        gauss_radau      = 2,
        middle_edge      = 3,
        superconvergent  = 4,
        equispaced       = 5,
        max_family       = 6
    } family_type; // update also family_name[] in quatration.cc

// allocators:

    quadrature_option(
        family_type ft = quadrature_option::gauss,
        size_type k = std::numeric_limits<size_type>::max());

    quadrature_option (const quadrature_option& qopt);
    quadrature_option& operator= (const quadrature_option& qopt);

// accessors & modifiers:

    size_t      get_order() const;
    family_type get_family() const;
    std::string get_family_name() const;
    void set_order (size_t r);
    void set_family (family_type type);
    void set_family (std::string name);
// data:
protected:
    family_type _family;
    size_t      _order;
};

```

4.18 tensor - a N*N tensor, N=1,2,3

(Source file: 'nfem/geo_element/tensor.h')

Synopsis

The `tensor` class defines a 3*3 tensor, as the value of a tensorial valued field. Basic algebra with scalars, vectors of \mathbb{R}^3 (i.e. the point class) and `tensor` objects are supported.

Implementation

```

template<class T>
class tensor_basic {
    public:

        typedef size_t size_type;
        typedef T      element_type;
        typedef T      float_type;

    // allocators:

        tensor_basic (const T& init_val = 0);
        tensor_basic (T x[3][3]);
        tensor_basic (const tensor_basic<T>& a);
        static tensor_basic<T> eye (size_type d = 3);

#ifdef _RHEOLEF_HAVE_STD_INITIALIZER_LIST
        tensor_basic (const std::initializer_list<std::initializer_list<T> >& il);
#endif // _RHEOLEF_HAVE_STD_INITIALIZER_LIST

    // affectation:

        tensor_basic<T>& operator= (const tensor_basic<T>& a);
        tensor_basic<T>& operator= (const T& val);

    // modifiers:

        void fill (const T& init_val);
        void reset ();
        void set_row    (const point_basic<T>& r, size_t i, size_t d = 3);
        void set_column (const point_basic<T>& c, size_t j, size_t d = 3);

    // accessors:

        T& operator()(size_type i, size_type j);
        const T& operator()(size_type i, size_type j) const;
        point_basic<T> row(size_type i) const;
        point_basic<T> col(size_type i) const;
        size_t nrow() const; // = 3, for template matrix compatibility
        size_t ncol() const;

    // inputs/outputs:

        std::ostream& put (std::ostream& s, size_type d = 3) const;
        std::istream& get (std::istream&);

    // algebra:

        bool operator== (const tensor_basic<T>&) const;
        bool operator!= (const tensor_basic<T>& b) const { return ! operator== (b);

```

```

    const tensor_basic<T>& operator+ () const { return *this; }
    tensor_basic<T> operator- () const;
    tensor_basic<T> operator+ (const tensor_basic<T>& b) const;
    tensor_basic<T> operator- (const tensor_basic<T>& b) const;
    tensor_basic<T> operator* (const tensor_basic<T>& b) const;
    tensor_basic<T> operator* (const T& k) const;
    tensor_basic<T> operator/ (const T& k) const;
    point_basic<T> operator* (const point_basic<T>&) const;
    point_basic<T> trans_mult (const point_basic<T>& x) const;

// metric and geometric transformations:

    T determinant (size_type d = 3) const;

// spectral:
    // eigenvalues & eigenvectors:
    // a = q*d*q^T
    // a may be symmetric
    // where q=(q1,q2,q3) are eigenvectors in rows (orthonormal matrix)
    // and d=(d1,d2,d3) are eigenvalues, sorted in decreasing order d1 >= d2
    // return d
    point_basic<T> eig (tensor_basic<T>& q, size_t dim = 3) const;
    point_basic<T> eig (size_t dim = 3) const;

    // singular value decomposition:
    // a = u*s*v^T
    // a can be unsymmetric
    // where u=(u1,u2,u3) are left pseudo-eigenvectors in rows (orthonormal matrix)
    // v=(v1,v2,v3) are right pseudo-eigenvectors in rows (orthonormal matrix)
    // and s=(s1,s2,s3) are eigenvalues, sorted in decreasing order s1 >= s2
    // return s
    point_basic<T> svd (tensor_basic<T>& u, tensor_basic<T>& v, size_t dim = 3) const;

// data:
    T _x[3][3];
};
typedef tensor_basic<Float> tensor;

// algebra (cont.)

template <class U>
point_basic<U> operator* (const point_basic<U>& yt, const tensor_basic<U>& a);
template <class U>
tensor_basic<U> trans (const tensor_basic<U>& a, size_t d = 3);
template <class U>
void prod (const tensor_basic<U>& a, const tensor_basic<U>& b, tensor_basic<U>& res,
           size_t di=3, size_t dj=3, size_t dk=3);
// tr(a) = a00 + a11 + a22
template <class U>

```

```

U tr (const tensor_basic<U>& a, size_t d=3);
template <class U>
U ddot (const tensor_basic<U>&, const tensor_basic<U>&);
// a = u otimes v <==> aij = ui*vj
template <class U>
tensor_basic<U> otimes (const point_basic<U>& u, const point_basic<U>& v, size_t d=3);
template <class U>
tensor_basic<U> inv (const tensor_basic<U>& a, size_t d = 3);
template <class U>
tensor_basic<U> diag (const point_basic<U>& d);
template <class U>
point_basic<U> diag (const tensor_basic<U>& a);
template <class U>
U determinant (const tensor_basic<U>& A, size_t d = 3);
template <class U>
bool invert_3x3 (const tensor_basic<U>& A, tensor_basic<U>& result);

// nonlinear algebra:
template<class T>
tensor_basic<T> exp (const tensor_basic<T>& a, size_t d = 3);

// inputs/outputs:
template<class T>
inline
std::istream& operator>> (std::istream& in, tensor_basic<T>& a)
{
    return a.get (in);
}
template<class T>
inline
std::ostream& operator<< (std::ostream& out, const tensor_basic<T>& a)
{
    return a.put (out);
}
// t += a otimes b
template<class T>
void cumul_otimes (tensor_basic<T>& t, const point_basic<T>& a, const point_basic<T>& b);
template<class T>
void cumul_otimes (tensor_basic<T>& t, const point_basic<T>& a, const point_basic<T>& b);

```

4.19 tensor3 - a third order tensor

(Source file: 'nfem/geo_element/tensor3.h')

Synopsis

The `tensor3` class defines a fourth tensor where indices varie from zero to 2 (aka 3D physical space).

Implementation

```

template<class T>
class tensor3_basic {
public:

    typedef size_t size_type;
    typedef T      element_type;
    typedef T      float_type;

    // allocators:

    tensor3_basic (const T& init_val = 0);
    tensor3_basic (const tensor3_basic<T>& a);

    // affectation:

    tensor3_basic<T>& operator= (const tensor3_basic<T>& a);
    tensor3_basic<T>& operator= (const T& val);

    // accessors:

    T&      operator()(size_type i, size_type j, size_type k);
    const T& operator()(size_type i, size_type j, size_type k) const;

    // algebra
    tensor3_basic<T>& operator*= (const T& k);
    tensor3_basic<T>& operator/= (const T& k) { return operator*= (1./k); }
    tensor3_basic<T>  operator*  (const T& k) const;
    tensor3_basic<T>  operator/  (const T& k) const;
    tensor3_basic<T>  operator*  (const point_basic<T>& v) const;
    tensor3_basic<T>  operator*  (const tensor_basic<T>& b) const;
    tensor3_basic<T>  operator+  (const tensor3_basic<T>& b) const;
    tensor3_basic<T>  operator-  (const tensor3_basic<T>& b) const;

    // inputs/outputs:

    std::ostream& put (std::ostream& s, size_type d = 3) const;
    std::istream& get (std::istream&);

    // data:
protected:
    T _x [3][3][3];
};
typedef tensor3_basic<Float> tensor3;

```


4.20 tensor4 - a fourth order tensor

(Source file: 'nfem/geo_element/tensor4.h')

Synopsis

The `tensor4` class defines a fourth tensor where indices varie from zero to 2 (aka 3D physical space).

Implementation

```
template<class T>
class tensor4_basic {
public:

    typedef size_t size_type;
    typedef T      element_type;
    typedef T      float_type;

    // allocators:

    tensor4_basic ();
    explicit tensor4_basic (const T& init_val);
    tensor4_basic (const tensor4_basic<T>& a);
    static tensor4_basic<T> eye (size_type d = 3);

#ifdef _RHEOLEF_HAVE_STD_INITIALIZER_LIST
        tensor4_basic (const std::initializer_list<std::initializer_list<
                        std::initializer_list<std::initializer_list<T> > > >&
#endif // _RHEOLEF_HAVE_STD_INITIALIZER_LIST

    // affectation:

    tensor4_basic<T>& operator= (const tensor4_basic<T>& a);
    tensor4_basic<T>& operator= (const T& val);

    // accessors:

    T&      operator()(size_type i, size_type j, size_type k, size_type l);
    const T& operator()(size_type i, size_type j, size_type k, size_type l) const;

    tensor4_basic<T>& operator()(size_type i, size_type j);
    const tensor4_basic<T>& operator()(size_type i, size_type j) const;

    // algebra:

    tensor4_basic<T>& operator*= (const T& k);
    tensor4_basic<T>& operator/= (const T& k) { return operator*= (1./k); }
    tensor4_basic<T> operator* (const T& k) const;
    tensor4_basic<T> operator/ (const T& k) const;
```

```

    tensor4_basic<T>  operator+ (const tensor4_basic<T>& b) const;
    tensor4_basic<T>  operator- (const tensor4_basic<T>& b) const;

// io:
    std::ostream& put (std::ostream& out, size_type d=3) const;

// data:
protected:
    tensor_basic<tensor_basic<T> > _x;
};
typedef tensor4_basic<Float> tensor4;

// nonlinear algebra:
template <class T>
T norm2 (const tensor4_basic<T>&);
template <class T>
T norm (const tensor4_basic<T>& a) { return sqrt(norm2(a)); }
template <class T>
tensor4_basic<T> dexp (const tensor_basic<T>& a, size_t d = 3);

// algebra:
template <class T>
tensor_basic<T> ddot (const tensor4_basic<T>&, const tensor_basic<T>&);
template <class T>
tensor_basic<T> ddot (const tensor_basic<T>&, const tensor4_basic<T>&);

```

4.21 asr - associative sparse matrix

(Source file: 'skit/plib2/asr.h')

Synopsis

Associative sparse matrix container, used during FEM assembling process.

Implementation note

Elements are stored row by row using the pair_set class bqsed on the STL map class.
Implementation of asr uses disarray<pair_set>

Implementation

```

template<class T, class M = rheo_default_memory_model, class A = std::allocator<T> >
class asr : public disarray<pair_set<T,A>, M, A> {
public:
// typedefs:

    typedef pair_set<T,A>                row_type;
    typedef disarray<row_type,M,A>       base;
    typedef typename base::size_type     size_type;

```

```

typedef M                                     memory_type;

struct dis_reference {
    dis_reference (typename base::dis_reference row_dis_i, size_type dis_j)
        : _row_dis_i(row_dis_i), _dis_j(dis_j) {}

    dis_reference& operator+= (const T& value) {
        _row_dis_i += std::pair<size_type,T>(_dis_j,value);
        return *this;
    }
    typename base::dis_reference _row_dis_i;
    size_type _dis_j;
};

// allocators/deallocators:

asr (const A& alloc = A())
    : base(distributor(), row_type(alloc), alloc), _col_ownership(), _nnz(0), _dis_j(0) {}

asr (const distributor& row_ownership, const distributor& col_ownership, const A& alloc = A())
    : base(row_ownership, row_type(alloc), alloc), _col_ownership(col_ownership), _nnz(0), _dis_j(0) {}

asr (const csr_rep<T,M>&, const A& alloc = A());
asr (const csr<T,M>&, const A& alloc = A());
void build_from_csr (const csr_rep<T,M>&);

void resize (const distributor& row_ownership, const distributor& col_ownership)
{
    base::resize (row_ownership);
    _col_ownership = col_ownership;
    _nnz = _dis_nnz = 0;
}

// accessors:

const communicator& comm() const { return base::comm(); }

size_type nrow () const { return base::size(); }
size_type ncol () const { return _col_ownership.size(); }
size_type nnz () const { return _nnz; }

size_type dis_nrow () const { return base::dis_size(); }
size_type dis_ncol () const { return _col_ownership.dis_size(); }
size_type dis_nnz () const { return _dis_nnz; }
const distributor& row_ownership() const { return base::ownership(); }
const distributor& col_ownership() const { return _col_ownership; }

// modifiers:

```

```

T operator()          (size_type      i, size_type dis_j) const;
T&      semi_dis_entry (size_type      i, size_type dis_j);
dis_reference dis_entry (size_type dis_i, size_type dis_j);

// dis_entry_assembly_end is redefined in order to recompute _nnz and _dis_nnz
void dis_entry_assembly_begin() { base::dis_entry_assembly_begin (pair_set_add_
void dis_entry_assembly_end()   { base::dis_entry_assembly_end   (pair_set_add_
void dis_entry_assembly()       { dis_entry_assembly_begin(); dis_entry_assembl

// io:
odiststream& put (odiststream& ops) const;
idiststream& get (idiststream& ips);

// internal:
odiststream& put_mpi          (odiststream& ops) const;
odiststream& put_seq          (odiststream& ops, size_type first_dis_i = 0
odiststream& put_seq_sparse_matlab (odiststream& ops, size_type first_dis_i = 0
odiststream& put_seq_matrix_market (odiststream& ops, size_type first_dis_i = 0
protected:
    void _recompute_nnz();
// data:
distributor _col_ownership;
size_type   _nnz;
size_type   _dis_nnz;
};

```

4.22 csr - compressed sparse row matrix

(Source file: 'skit/plib2/csr.h')

Synopsis

Distributed compressed sparse matrix container stored row by row.

Description

Sparse matrix are compressed by rows. In distributed environment, the distribution follows the row distributor (see [\[distributor class\]](#), page [\[undefined\]](#)).

Algebra

Adding or subtracting two matrices writes $\mathbf{a}+\mathbf{b}$ and $\mathbf{a}-\mathbf{b}$, respectively, and multiplying a matrix by a scalar writes $\lambda\mathbf{x}$. Thus, any linear combination of sparse matrices is available.

Matrix-vector product writes $\mathbf{a}\mathbf{x}$ where \mathbf{x} is a vector (see [\[vec class\]](#), page [\[undefined\]](#)).

Limitations

Some basic linear algebra is still under development: `a.trans_mult(x)` matrix transpose vector product, `trans(a)` matrix transpose, `a*b` matrix product.

Implementation

```
template<class T>
class csr<T,sequential> : public smart_pointer<csr_rep<T,sequential> > {
public:

// typedefs:

    typedef csr_rep<T,sequential>          rep;
    typedef smart_pointer<rep>              base;
    typedef typename rep::memory_type      memory_type;
    typedef typename rep::size_type        size_type;
    typedef typename rep::element_type     element_type;
    typedef typename rep::iterator         iterator;
    typedef typename rep::const_iterator   const_iterator;
    typedef typename rep::data_iterator    data_iterator;
    typedef typename rep::const_data_iterator const_data_iterator;

// allocators/deallocators:

    csr() : base(new_macro(rep())) {}
    template<class A>
    explicit csr(const asr<T,sequential,A>& a) : base(new_macro(rep(a))) {}
    void resize (size_type loc_nrow1 = 0, size_type loc_ncol1 = 0, size_type loc_nnz1)
        { base::data().resize(loc_nrow1, loc_ncol1, loc_nnz1); }
    void resize (const distributor& row_ownership, const distributor& col_ownership,
        { base::data().resize(row_ownership, col_ownership, nnz1); }

// allocators from initializer list (c++ 2011):

#ifdef _RHEOLEF_HAVE_STD_INITIALIZER_LIST
    csr (const std::initializer_list<csr_concat_value<T,sequential> >& init_list);
    csr (const std::initializer_list<csr_concat_line<T,sequential> >& init_list);
#endif // _RHEOLEF_HAVE_STD_INITIALIZER_LIST

// accessors:

    // global sizes
    const distributor& row_ownership() const { return base::data().row_ownership(); }
    const distributor& col_ownership() const { return base::data().col_ownership(); }
    size_type dis_nrow () const { return row_ownership().dis_size(); }
    size_type dis_ncol () const { return col_ownership().dis_size(); }
    size_type dis_nnz () const { return base::data().nnz(); }
    size_type dis_ext_nnz () const { return 0; }
    bool is_symmetric() const { return base::data().is_symmetric(); }
```

```

void set_symmetry (bool is_symm) const    { base::data().set_symmetry(is_symm); }
void set_symmetry_by_check (const T& tol = std::numeric_limits<T>::epsilon()) const
                                         { base::data().set_symmetry_by_check(tol); }
bool is_definite_positive() const        { return base::data().is_definite_positive(); }
void set_definite_positive (bool is_defpos) const { base::data().set_definite_positive(is_defpos); }
size_type pattern_dimension() const       { return base::data().pattern_dimension(); }
void set_pattern_dimension(size_type dim) const { base::data().set_pattern_dimension(dim); }
T max_abs () const                        { return base::data().max_abs(); }

// local sizes
size_type nrow () const                   { return base::data().nrow(); }
size_type ncol () const                   { return base::data().ncol(); }
size_type nnz () const                    { return base::data().nnz(); }

// range on local memory
size_type row_first_index () const        { return base::data().row_first_index(); }
size_type row_last_index () const         { return base::data().row_last_index(); }
size_type col_first_index () const        { return base::data().col_first_index(); }
size_type col_last_index () const         { return base::data().col_last_index(); }

const_iterator begin() const              { return base::data().begin(); }
const_iterator end() const                { return base::data().end(); }
iterator begin_nonconst()                 { return base::data().begin(); }
iterator end_nonconst()                   { return base::data().end(); }

// accessors, only for distributed (for interface compatibility)
size_type ext_nnz() const                  { return 0; }
const_iterator ext_begin() const           { return const_iterator(); }
const_iterator ext_end() const             { return const_iterator(); }
iterator ext_begin_nonconst()              { return iterator(); }
iterator ext_end_nonconst()                { return iterator(); }
size_type jext2dis_j (size_type jext) const { return 0; }

// algebra:

// y := a*x
void mult (const vec<element_type,sequential>& x, vec<element_type,sequential>& y) const
{
    base::data().mult (x,y);
}
vec<element_type,sequential> operator* (const vec<element_type,sequential>& x,
    vec<element_type,sequential> y (row_ownership(), element_type()));
{
    mult (x, y);
    return y;
}
void trans_mult (const vec<element_type,sequential>& x, vec<element_type,sequential>& y) const
{
    base::data().trans_mult (x,y);
}
vec<element_type,sequential> trans_mult (const vec<element_type,sequential>& x,
    vec<element_type,sequential> y (col_ownership(), element_type()));

```

```

        trans_mult (x, y);
        return y;
    }
    // a+b, a-b, a*b
    csr<T,sequential> operator+ (const csr<T,sequential>& b) const;
    csr<T,sequential> operator- (const csr<T,sequential>& b) const;
    csr<T,sequential> operator* (const csr<T,sequential>& b) const;

    // lambda*a
    csr<T,sequential>& operator*= (const T& lambda) {
        base::data().operator*= (lambda);
        return *this;
    }
    // output:

    void dump (const std::string& name) const { base::data().dump(name); }
};
// lambda*a
template<class T>
inline
csr<T,sequential>
operator* (const T& lambda, const csr<T,sequential>& a)
{
    csr<T,sequential> b = a;
    b.operator*= (lambda);
    return b;
}
// -a
template<class T>
inline
csr<T,sequential>
operator- (const csr<T,sequential>& a)
{
    return T(-1)*a;
}
// trans(a)
template<class T>
inline
csr<T,sequential>
trans (const csr<T,sequential>& a)
{
    csr<T,sequential> b;
    a.data().build_transpose (b.data());
    return b;
}

```

Implementation

```

template<class T>

```

```

class csr<T,distributed> : public smart_pointer<csr_rep<T,distributed> > {
public:

// typedefs:

    typedef csr_rep<T,distributed>                rep;
    typedef smart_pointer<rep>                    base;
    typedef typename rep::memory_type             memory_type;
    typedef typename rep::size_type               size_type;
    typedef typename rep::element_type            element_type;
    typedef typename rep::iterator                iterator;
    typedef typename rep::const_iterator          const_iterator;
    typedef typename rep::data_iterator           data_iterator;
    typedef typename rep::const_data_iterator     const_data_iterator;

// allocators/deallocators:

    csr() : base(new_macro(rep())) {}
    template<class A>
    explicit csr(const asr<T,memory_type,A>& a) : base(new_macro(rep(a))) {}
    void resize (const distributor& row_ownership, const distributor& col_ownership
        { base::data().resize(row_ownership, col_ownership, nnz1); }

// allocators from initializer list (c++ 2011):

#ifdef _RHEOLEF_HAVE_STD_INITIALIZER_LIST
    csr (const std::initializer_list<csr_concat_value<T,distributed> >& init_list);
    csr (const std::initializer_list<csr_concat_line<T,distributed> >& init_list);
#endif // _RHEOLEF_HAVE_STD_INITIALIZER_LIST

// accessors:

    // global sizes
    const distributor& row_ownership() const { return base::data().row_ownership(); }
    const distributor& col_ownership() const { return base::data().col_ownership(); }
    size_type dis_nrow () const { return row_ownership().dis_size(); }
    size_type dis_ncol () const { return col_ownership().dis_size(); }
    size_type dis_nnz () const { return base::data().dis_nnz(); }
    size_type dis_ext_nnz () const { return base::data().dis_ext_nnz(); }
    bool is_symmetric() const { return base::data().is_symmetric(); }
    void set_symmetry (bool is_symm) const { base::data().set_symmetry(is_symm); }
    void set_symmetry_by_check (const T& tol = std::numeric_limits<T>::epsilon()) const
        { base::data().set_symmetry_by_check(tol); }
    bool is_definite_positive() const { return base::data().is_definite_posi; }
    void set_definite_positive (bool is_defpos) const { base::data().set_definite_p; }
    size_type pattern_dimension() const { return base::data().pattern_dimension; }
    void set_pattern_dimension(size_type dim) const { base::data().set_pattern_dime; }
    T max_abs () const { return base::data().max_abs(); }

```



```

// local sizes
size_type nrow () const { return base::data().nrow(); }
size_type ncol () const { return base::data().ncol(); }
size_type nnz () const { return base::data().nnz(); }

// range on local memory
size_type row_first_index () const { return base::data().row_first_index(); }
size_type row_last_index () const { return base::data().row_last_index(); }
size_type col_first_index () const { return base::data().col_first_index(); }
size_type col_last_index () const { return base::data().col_last_index(); }

const_iterator begin() const { return base::data().begin(); }
const_iterator end() const { return base::data().end(); }
iterator begin_nonconst() { return base::data().begin(); }
iterator end_nonconst() { return base::data().end(); }

// accessors, only for distributed
size_type ext_nnz() const { return base::data().ext_nnz(); }
const_iterator ext_begin() const { return base::data().ext_begin(); }
const_iterator ext_end() const { return base::data().ext_end(); }
iterator ext_begin_nonconst() { return base::data().ext_begin(); }
iterator ext_end_nonconst() { return base::data().ext_end(); }
size_type jext2dis_j (size_type jext) const { return base::data().jext2dis_j(jext); }

// algebra:

// y := a*x
void mult (const vec<element_type,distributed>& x, vec<element_type,distributed>
    base::data().mult (x,y);
}
vec<element_type,distributed> operator* (const vec<element_type,distributed>& x
    vec<element_type,distributed> y (row_ownership(), element_type());
    mult (x, y);
    return y;
}
void trans_mult (const vec<element_type,distributed>& x, vec<element_type,distributed>
    base::data().trans_mult (x,y);
}
vec<element_type,distributed> trans_mult (const vec<element_type,distributed>&
    vec<element_type,distributed> y (col_ownership(), element_type());
    trans_mult (x, y);
    return y;
}
// a+b, a-b, a*b
csr<T,distributed> operator+ (const csr<T,distributed>& b) const;
csr<T,distributed> operator- (const csr<T,distributed>& b) const;
csr<T,distributed> operator* (const csr<T,distributed>& b) const;

```

```

        // lambda*a
        csr<T,distributed>& operator*= (const T& lambda) {
            base::data().operator*= (lambda);
            return *this;
        }
    // output:

        void dump (const std::string& name) const { base::data().dump(name); }
};
// lambda*a
template<class T>
inline
csr<T,distributed>
operator* (const T& lambda, const csr<T,distributed>& a)
{
    csr<T,distributed> b = a;
    b.operator*= (lambda);
    return b;
}
// -a
template<class T>
inline
csr<T,distributed>
operator- (const csr<T,distributed>& a)
{
    return T(-1)*a;
}
// trans(a)
template<class T>
inline
csr<T,distributed>
trans (const csr<T,distributed>& a)
{
    csr<T,distributed> b;
    a.data().build_transpose (b.data());
    return b;
}
#endif // _RHEOLEF_HAVE_MPI

// b = f(a); f as a class-function or usual fct
template<class T, class M, class Function>
csr<T,M>
apply (Function f, const csr<T,M>& a)
{
    csr<T,M> b = a;
    typename csr<T,M>::size_type n = a.nrow();
    typename csr<T,M>::const_iterator dia_ia = a.begin();
    typename csr<T,M>::iterator dia_ib = b.begin_nonconst();
    pair_transform_second (dia_ia[0], dia_ia[n], dia_ib[0], f);
}

```

```

    if (a.ext_nnz() != 0) {
        typename csr<T,M>::const_iterator ext_ia = a.ext_begin();
        typename csr<T,M>::iterator      ext_ib = b.ext_begin_nonconst();
        pair_transform_second (ext_ia[0], ext_ia[n], ext_ib[0], f);
    }
    return b;
}
template<class T, class M, class Function>
csr<T,M>
apply (T (*f)(const T&), const csr<T,M>& a)
{
    return apply (std::ptr_fun(f), a);
}

```

4.23 dia - diagonal matrix

(Source file: 'skit/plib2/dia.h')

Description

The class implements a diagonal matrix. A declaration without any parameters correspond to a null size matrix:

```
dia<Float> d;
```

The constructor can be invoked with a *ownership* parameter (see [\[distributor class\]](#), page [\[undefined\]](#)):

```
dia<Float> d(ownership);
```

or an initialiser, either a vector (see [\[vec class\]](#), page [\[undefined\]](#)):

```
dia<Float> d(v);
```

or a csr matrix (see [\[csr class\]](#), page [\[undefined\]](#)):

```
dia<Float> d(a);
```

The conversion from *dia* to *vec* or *csr* is explicit.

When a diagonal matrix is constructed from a *csr* matrix, the definition of the diagonal of matrix is *always* a vector of size *row_ownership* which contains the elements in rows 1 to *nrow* of the matrix that are contained in the diagonal. If the diagonal element falls outside the matrix, i.e. *ncol* < *nrow* then it is defined as a zero entry.

Preconditioner interface

The class presents a preconditioner interface, as the [\[solver class\]](#), page [\[undefined\]](#), so that it can be used as preconditioner to the iterative solvers suite (see [\[cg algorithm\]](#), page [\[undefined\]](#)).

Implementation

```
template<class T, class M = rheo_default_memory_model>
class dia : public vec<T,M> {
public:

    // typedefs:

        typedef typename vec<T,M>::size_type      size_type;
        typedef typename vec<T,M>::iterator       iterator;
        typedef typename vec<T,M>::const_iterator const_iterator;

    // allocators/deallocators:

        explicit dia (const distributor& ownership = distributor(),
                      const T& init_val = std::numeric_limits<T>::max());

        explicit dia (const vec<T,M>& u);
        explicit dia (const csr<T,M>& a);
        dia<T,M>& operator= (const T& lambda);

    // preconditionner interface: solves d*x=b

        vec<T,M> solve (const vec<T,M>& b) const;
        vec<T,M> trans_solve (const vec<T,M>& b) const;
};
template <class T, class M>
dia<T,M> operator/ (const T& lambda, const dia<T,M>& d);

template <class T, class M>
vec<T,M> operator* (const dia<T,M>& d, const vec<T,M>& x);
```

4.24 disarray - container in distributed environment

(Source file: 'skit/plib2/disarray.h')

Synopsis

STL-like vector container for a distributed memory machine model.

Example

A sample usage of the class is:

```
int main(int argc, char**argv) {
    environment distributed(argc, argv);
    disarray<double> x(distributor(100), 3.14);
    dout << x << endl;
}
```

The `disarray<T>` interface is similar to those of the `std::vector<T>` with the addition of some communication features in the distributed case: write accesses with `entry/assembly` and read access with `dis_at`.

Distributed write access

Loop on any `dis_i` that is not managed by the current processor:

```
x.dis_entry (dis_i) = value;
```

and then, after loop, perform all communication:

```
x.dis_entry_assembly();
```

After this command, each value is stored in the `disarray`, available the processor associated to `dis_i`.

Distributed read access

First, define the set of indexes:

```
std::set<size_t> ext_idx_set;
```

Then, loop on `dis_i` indexes that are not managed by the current processor:

```
ext_idx_set.insert (dis_i);
```

After the loop, performs the communications:

```
x.set_dis_indexes (ext_idx_set);
```

After this command, each values associated to the `dis_i` index, and that belongs to the index set, is now available also on the current processor as:

```
value = x.dis_at (dis_i);
```

For convenience, if `dis_i` is managed by the current processor, this function returns also the value.

Note

The class takes two template parameters: one for the type `T` and the second for the memory model `M`, that could be either `M=distributed` or `M=sequential`. The two cases are associated to two different implementations, but proposes exactly the same interface. The sequential interface propose also a supplementary constructor:

```
disarray<double,sequential> x(local_size, init_val);
```

This constructor is a STL-like one but could be consufused in the distributed case, since there are two sizes: a local one and a global one. In that case, the use of the distributor, as a generalization of the size concept, clarify the situation (see `<undefined>` [distributor class], page `<undefined>`).

Implementation note

"scatter" via "get_dis_entry".

"gather" via "dis_entry(dis_i) = value" or "dis_entry(dis_i) += value". Note that += applies when $T = \text{idx_set}$ where idx_set is a wrapper class of `std::set<size_t>`; the += operator represents the union of a set. The operator= is used when $T = \text{double}$ or others simple T types without algebra. If there is a conflict, i.e. several processes set the `dis_i` index, then the result of operator+= depends upon the order of the process at each run and is not deterministic. Such ambiguous behavior is not detected yet at run time.

Implementation

```
template <class T, class A>
class disarray<T,sequential,A> : public smart_pointer<disarray_rep<T,sequential,A> >
public:

    // typedefs:

        typedef disarray_rep<T,sequential,A>      rep;
        typedef smart_pointer<rep>                 base;

        typedef sequential                        memory_type;
        typedef typename rep::size_type           size_type;
        typedef typename rep::difference_type      difference_type;
        typedef typename rep::value_type           value_type;
        typedef typename rep::reference            reference;
        typedef typename rep::dis_reference        dis_reference;
        typedef typename rep::iterator             iterator;
        typedef typename rep::const_reference      const_reference;
        typedef typename rep::const_iterator      const_iterator;

    // allocators:

        disarray      (size_type loc_size = 0,          const T& init_val = T(), const A&
        void resize (size_type loc_size = 0,          const T& init_val = T());
        disarray      (const distributor& ownership, const T& init_val = T(), const A&
        void resize (const distributor& ownership, const T& init_val = T());

    // local accessors & modifiers:

        A get_allocator() const { return base::data().get_allocator(); }
        size_type size () const { return base::data().size(); }
        size_type dis_size () const { return base::data().dis_size(); }
        const distributor& ownership() const { return base::data().ownership(); }
        const communicator& comm() const { return ownership().comm(); }

        reference      operator[] (size_type i) { return base::data().operator[]
        const_reference operator[] (size_type i) const { return base::data().operator[]
```

```

reference      operator() (size_type i)      { return base::data().operator[]
const_reference operator() (size_type i) const { return base::data().operator[]
const_reference dis_at (size_type dis_i) const { return operator[] (dis_i); }

        iterator begin()      { return base::data().begin(); }
const_iterator begin() const { return base::data().begin(); }
        iterator end()        { return base::data().end(); }
const_iterator end() const    { return base::data().end(); }

// global modifiers (for compatibility with distributed interface):

dis_reference dis_entry (size_type dis_i) { return base::data().dis_entry(dis_i)
template<class SetOp = typename default_set_op<T>::type>
void dis_entry_assembly (SetOp my_set_op = SetOp()) {}
template<class SetOp = typename default_set_op<T>::type>
void dis_entry_assembly_begin (SetOp my_set_op = SetOp()) {}
template<class SetOp = typename default_set_op<T>::type>
void dis_entry_assembly_end (SetOp my_set_op = SetOp()) {}

void dis_entry_assembly_begin() {}
void dis_entry_assembly_end()   {}
void dis_entry_assembly()       {}

void reset_dis_indexes() const {}
template<class Set> void set_dis_indexes (const Set& ext_idx_set) const {}
template<class Set> void append_dis_indexes (const Set& ext_idx_set) const {}
template<class Set, class Map> void append_dis_entry (const Set& ext_idx_set, M
template<class Set, class Map> void get_dis_entry (const Set& ext_idx_set, M

// apply a partition:

template<class RepSize>
void repartition (
    const RepSize&      partition,          // old_numbering for *this
    disarray<T,sequential,A>& new_disarray,  // old_ownership
    RepSize&            old_numbering,      // new_ownership (created)
    RepSize&            new_numbering) const // new_ownership
{ return base::data().repartition (partition, new_disarray, old_numbering,

template<class RepSize>
void permutation_apply (
    const RepSize&      new_numbering,      // old_numbering for *this
    disarray<T,sequential,A>& new_disarray) const // old_ownership
{ return base::data().permutation_apply (new_numbering, new_disarray); }

void reverse_permutation (
    disarray<size_type,sequential,A>& inew2dis_iold) const // old_ownership for
{ base::data().reverse_permutation (inew2dis_iold.data()); }

```

```

// i/o:

    odistream& put_values (odistream& ops) const { return base::data().put_valu
    idistream& get_values (idistream& ips)          { return base::data().get_valu
    template <class GetFunction>
    idistream& get_values (idistream& ips, GetFunction get_element)          { ret
    template <class PutFunction>
    odistream& put_values (odistream& ops, PutFunction put_element) const { ret
    void dump (std::string name) const { return base::data().dump(name); }
};

```

Implementation

```

template <class T, class A>
class disarray<T,distributed,A> : public smart_pointer<disarray_rep<T,distributed,A>
public:

// typedefs:

    typedef disarray_rep<T,distributed,A>      rep;
    typedef smart_pointer<rep>                  base;

    typedef distributed                        memory_type;
    typedef typename rep::size_type           size_type;
    typedef typename rep::difference_type     difference_type;
    typedef typename rep::value_type          value_type;
    typedef typename rep::reference            reference;
    typedef typename rep::dis_reference       dis_reference;
    typedef typename rep::iterator            iterator;
    typedef typename rep::const_reference     const_reference;
    typedef typename rep::const_iterator     const_iterator;
    typedef typename rep::scatter_map_type    scatter_map_type;

// allocators:

    disarray      (const distributor& ownership = distributor(), const T& init_val
    void resize (const distributor& ownership = distributor(), const T& init_val =

// local accessors & modifiers:

    A get_allocator() const                { return base::data().get_allocator(); }
    size_type      size () const            { return base::data().size(); }
    size_type dis_size () const            { return base::data().dis_size(); }
    const distributor& ownership() const    { return base::data().ownership(); }
    const communicator& comm() const       { return base::data().comm(); }

    reference      operator[] (size_type i)      { return base::data().operator[]
    const_reference operator[] (size_type i) const { return base::data().operator[]
    reference      operator() (size_type i)      { return base::data().operator[]

```



```

const_reference operator() (size_type i) const { return base::data().operator[]

        iterator begin()      { return base::data().begin(); }
const_iterator begin() const { return base::data().begin(); }
        iterator end()        { return base::data().end(); }
const_iterator end() const   { return base::data().end(); }

// global accessor:

template<class Set, class Map>
void append_dis_entry (const Set& ext_idx_set, Map& ext_idx_map) const { base::

template<class Set, class Map>
void get_dis_entry    (const Set& ext_idx_set, Map& ext_idx_map) const { base::

template<class Set>
void append_dis_indexes (const Set& ext_idx_set) const { base::data().append_di
void reset_dis_indexes() const { base::data().reset_dis_indexes(); }

template<class Set>
void set_dis_indexes    (const Set& ext_idx_set) const { base::data().set_dis_i

const T& dis_at (size_type dis_i) const { return base::data().dis_at (dis_i); }

// get all external pairs (dis_i, values):
const scatter_map_type& get_dis_map_entries() const { return base::data().get_d

// global modifiers (for compatibility with distributed interface):

dis_reference dis_entry (size_type dis_i) { return base::data().dis_entry(dis_i

template<class SetOp = typename default_set_op<T>::type>
void dis_entry_assembly_begin (SetOp my_set_op = SetOp()) { base::data().dis_en
template<class SetOp = typename default_set_op<T>::type>
void dis_entry_assembly_end   (SetOp my_set_op = SetOp()) { base::data().dis_en
template<class SetOp = typename default_set_op<T>::type>
void dis_entry_assembly      (SetOp my_set_op = SetOp()) { base::data().dis_en

void dis_entry_assembly_begin() { base::data().template dis_entry_assembly_begi
void dis_entry_assembly_end()   { base::data().template dis_entry_assembly_end<
void dis_entry_assembly()       { dis_entry_assembly_begin(); dis_entry_assembl

// apply a partition:

template<class RepSize>
void repartition (
    const RepSize&          partition,          // old_numbering for *this
    disarray<T,distributed>& new_disarray,       // old_ownership
    RepSize&                old_numbering,       // new_ownership (created

```

```

        RepSize& new_numbering) const // old_ownership
    { return base::data().repartition (partition.data(), new_disarray.data(), o

template<class RepSize>
void permutation_apply ( // old_numbering for *this
    const RepSize& new_numbering, // old_ownership
    disarray<T,distributed,A>& new_disarray) const // new_ownership (already
    { base::data().permutation_apply (new_numbering.data(), new_disarray.data()

void reverse_permutation ( // old_ownership for
    disarray<size_type,distributed,A>& inew2dis_iold) const // new_ownership
    { base::data().reverse_permutation (inew2dis_iold.data()); }

// i/o:

odiststream& put_values (odiststream& ops) const { return base::data().put_valu
idiststream& get_values (idiststream& ips) { return base::data().get_valu
void dump (std::string name) const { return base::data().dump(name); }

template <class GetFunction>
idiststream& get_values (idiststream& ips, GetFunction get_element) { ret
template <class PutFunction>
odiststream& put_values (odiststream& ops, PutFunction put_element) const { ret
template <class PutFunction, class A2> odiststream& permuted_put_values (
    odiststream& ops, const disarray<size_type,distributed,A2>& perm, P
    { return base:
};

```

4.25 distributor - data distribution table

(Source file: 'skit/plib2/distributor.h')

Synopsis

Used by "disarray"(1), "asr"(1) and "csr"(1). and such classes that distribute data as chunk.

Implementation

```

class distributor : public Vector<std::allocator<int>::size_type> {
public:

    // typedefs:

    typedef std::allocator<int>::size_type size_type;
    typedef Vector<size_type> _base;
    typedef _base::iterator iterator;
    typedef _base::const_iterator const_iterator;
    typedef int tag_type;

```

```

        typedef communicator                communicator_type;

// constants:

        static const size_type decide = size_type(-1);

// allocators/deallocators:

        distributor(
            size_type dis_size = 0,
            const communicator_type& c = communicator_type(),
            size_type loc_size = decide);

        distributor(const distributor&);
        ~distributor();

        void resize(
            size_type dis_size = 0,
            const communicator_type& c = communicator_type(),
            size_type loc_size = decide);

// accessors:

        const communicator_type& comm() const;

        /// global and local sizes
        size_type dis_size () const;

        /// current process id
        size_type process () const;

        /// number of processes
        size_type n_process () const;

        /// find iproc associated to a global index dis_i: CPU=log(nproc)
        size_type find_owner (size_type dis_i) const;

        /// global index range and local size owned by ip-th process
        size_type first_index (size_type ip) const;
        size_type last_index (size_type ip) const;
        size_type size (size_type ip) const;

        /// global index range and local size owned by current process
        size_type first_index () const;
        size_type last_index () const;
        size_type size () const;

        /// true when dis_i in [first_index(ip):last_index(ip)]
        bool is_owned (size_type dis_i, size_type ip) const;

```

```

        // the same with ip=current process
        bool is_owned (size_type dis_i) const;

        /// returns a new tag
        static tag_type get_new_tag();

    // comparators:

        bool operator== (const distributor&) const;
        bool operator!= (const distributor&) const;
    // data:
protected:
        communicator_type _comm;
};

```

4.26 idiststream, odiststream - distributed interface for large data streams

(Source file: 'skit/plib2/diststream.h')

Description

This class provides a parallel and distributed stream interface for large data management. File decompression is assumed using `gzip` and a recursive search in a directory list is provided for input.

```
odiststream foo("NAME", "suffix");
```

is similar

```
ofstream foo("NAME.suffix").
```

The main difference is that writing to `ofstream` is executed on all processes while `odiststream` manage nicely the multi-process environment. For convenience, the standard streams `cin`, `cout` and `cerr` are extended to `din`, `dout` and `derr`. Notice that, as `orheostream` (see [\[rheostream class\]](#), page [\[undefined\]](#)), if *NAME* does not end with `'suffix'`, then `'suffix'` is automatically added. By default, compression is performed on the fly with `gzip`, adding an additional `'gz'` suffix. The `flush` action is nicely handled in compression mode:

```
foo.flush();
```

This feature allows intermediate results to be available during long computations. The compression can be deactivated while opening a file by an optional argument:

```
odiststream foo("NAME", "suffix", io::nogz);
```

An existing compressed file can be reopen in **append** mode: new results will be appended at the end of an existing file:

```
odiststream foo("NAME", "suffix", io::app);
```

Conversely,

```
irheostream foo("NAME","suffix");
```

is similar to

```
ifstream foo("NAME.suffix").
```

Also `idiststream` manage nicely the multi-process environment. However, we look at a search path environment variable `RHEOPATH` in order to find `NAME` while suffix is assumed. Moreover, `gzip` compressed files, ending with the ‘.gz’ suffix is assumed, and decompression is done.

Implementation

```
class odiststream {
public:
    typedef std::size_t size_type;

    // allocators/deallocators:

    odiststream();
    odiststream (std::string filename, std::string suffix = "",
                 io::mode_type mode = io::out, const communicator& comm = communicator());
    odiststream (std::string filename,
                 io::mode_type mode, const communicator& comm = communicator());
    odiststream (std::string filename, std::string suffix, const communicator& comm);
    odiststream (std::string filename, const communicator& comm);
    odiststream(std::ostream& os, const communicator& comm = communicator());
    ~odiststream();

    // modifiers:

    void open (std::string filename, std::string suffix = "",
              io::mode_type mode = io::out, const communicator& comm = communicator());
    void open (std::string filename,
              io::mode_type mode, const communicator& comm = communicator());
    void open (std::string filename, std::string suffix,
              const communicator& comm);
    void open (std::string filename, const communicator& comm);
    void flush();
    void close();

    // accessors:

    const communicator& comm() const { return _comm; }
    bool good() const;
    operator bool() const { return good(); }
```

```

        static size_type io_proc();

// internals:

        std::ostream& os();
        bool nop();

protected:
// data:
        std::ostream* _ptr_os;
        bool          _use_alloc;
        communicator _comm;
};

```

Implementation

```

class idiststream {
public:
    typedef std::size_t size_type;

// allocators/deallocators:

    idiststream();
    idiststream (std::istream& is, const communicator& comm = communicator());
    idiststream (std::string filename, std::string suffix = "",
                 const communicator& comm = communicator());
    ~idiststream();

// modifiers:

    void open (std::string filename, std::string suffix = "",
              const communicator& comm = communicator());
    void close();

// accessors:

    const communicator& comm() const { return _comm; }
    bool good() const;
    operator bool() const { return good(); }
    static size_type io_proc();

// internals:

    std::istream& is();
    bool nop();
    bool do_load();

protected:
// data:

```

```

    std::istream* _ptr_is;
    bool          _use_alloc;
    communicator  _comm;
};

```

4.27 environment - rheolef initialization

(Source file: 'skit/plib2/environment.h')

Synopsis

This class is mainly used to initialize the MPI library: it should be called just after the `main(argc,argv)` declaration as in:

```

#include "rheolef.h"
using namespace rheolef;
int main(int argc, char**argv) {
    environment rheolef (argc, argv);
    ...
}

```

An optional third argument of the `environment` constructor allows one to set the MPI thread feature. By default, its value is `MPI_THREAD_SINGLE`, as defined in `mpi.h`. Other possible values are related to MPI multi-threaded, see the MPI documentation for more details.

4.28 eye - the identity matrix

(Source file: 'skit/plib2/eye.h')

Description

Following octave, the name `eye_rep` class is used in place of `I` to denote identity matrices because `I` is often used as a subscript or as `sqrt(-1)`. The dimensions of `eye_rep` are determined by the context. This class is here useful in the context of preconditioner interfaces: it allows calls of algorithms without any preconditioners, e.g.

```
int status = cg (a, x, b, eye_rep<Float>(), 100, 1e-7);
```

Implementation

```

template<class T, class M = rheo_default_memory_model>
class eye_rep : public solver_abstract_rep<T,M> {
public:
    eye_rep (const solver_option& opt = solver_option());
    void update_values (const csr<T,M>&) {}
    vec<T,M> operator* (const vec<T,M>& x) const { return x; }
    vec<T,M> solve      (const vec<T,M>& x) const { return x; }
    vec<T,M> trans_solve (const vec<T,M>& x) const { return x; }
};

```

4.29 solver - direct or interactive solver interface

(Source file: 'skit/plib2/solver.h')

Description

The class implements a matrix factorization: LU factorization for an unsymmetric matrix and Choleski factorisation for a symmetric one.

Let a be a square invertible matrix in `csr` format (see [\[csr class\]](#), page [\[undefined\]](#))).

```
csr<Float> a;
```

We get the factorization by:

```
solver sa (a);
```

Each call to the direct solver for $a*x = b$ writes either:

```
vec<Float> x = sa.solve(b);
```

When the matrix is modified in a computation loop but conserves its sparsity pattern, an efficient re-factorization writes:

```
sa.update_values (new_a);
x = sa.solve(b);
```

Automatic choice and customization

The symmetry of the matrix is tested via the `a.is_symmetric()` property (see [\[csr class\]](#), page [\[undefined\]](#)) while the choice between direct or iterative solver is switched by default from the `a.pattern_dimension()` value. When the pattern is 3D, an iterative method is faster and less memory consuming. Otherwise, for 1D or 2D problems, the direct method is preferred.

These default choices can be superseded by using explicit options:

```
solver_option opt;
opt.iterative = true;
solver sa (a, opt);
```

When using an iterative method, the `sa.solve(b)` call the conjugate gradient when the matrix is symmetric, or the generalized minimum residual algorithm when the matrix is unsymmetric.

Computation of the determinant

When using a direct method, the determinant of the `a` matrix can be computed as:

```
solver_option opt;
opt.iterative = false;
solver sa (a, opt);
```



```
cout << sa.det().mantissa << "*2^" << sa.det().exponent << endl;
```

The `sa.det()` method returns an object of type `solver::determinant_type` that contains a mantissa and an exponent in base 2. This feature is useful e.g. when tracking a change of sign in the determinant of a matrix.

Preconditionners for iterative solver

When using an iterative method, the default is to do no preconditioning. A suitable preconditionner can be supplied via:

```
solver_option opt;
opt.iterative = true;
solver sa (a, opt);
sa.set_preconditionner (pa);
x = sa.solve(b);
```

The supplied `pa` variable is also of type `solver`. A library of commonly used preconditionners is still in development.

Implementation

```
template <class T, class M = rheo_default_memory_model>
class solver_basic : public smart_pointer<solver_rep<T,M> > {
public:
// typedefs:

    typedef solver_rep<T,M> rep;
    typedef smart_pointer<rep> base;
    typedef typename rep::size_type size_type;
    typedef typename rep::determinant_type determinant_type;

// allocator:

    solver_basic ();
    explicit solver_basic (const csr<T,M>& a, const solver_option& opt = solver_option(0));
    const solver_option& option() const;
    void update_values (const csr<T,M>& a);
    void set_preconditionner (const solver_basic<T,M>&);

// accessors:

    vec<T,M> trans_solve (const vec<T,M>& b) const;
    vec<T,M> solve (const vec<T,M>& b) const;
    determinant_type det() const;
};
// factorizations:
template <class T, class M>
solver_basic<T,M> ldlt(const csr<T,M>& a, const solver_option& opt = solver_option(0));
```

```

template <class T, class M>
solver_basic<T,M> lu (const csr<T,M>& a, const solver_option& opt = solver_option(

// preconditionners:
template <class T, class M = rheo_default_memory_model>
solver_basic<T,M> eye_basic();
inline solver_basic<Float> eye() { return eye_basic<Float>(); }
template <class T, class M>
solver_basic<T,M> ic0 (const csr<T,M>& a, const solver_option& opt = solver_option(
template <class T, class M>
solver_basic<T,M> ilu0(const csr<T,M>& a, const solver_option& opt = solver_option(
template <class T, class M>
solver_basic<T,M> ldlt_seq(const csr<T,M>& a, const solver_option& opt = solver_opt
template <class T, class M>
solver_basic<T,M> lu_seq (const csr<T,M>& a, const solver_option& opt = solver_opt

typedef solver_basic<Float> solver;

```

4.30 solver_abtb – direct or iterative solver interface for mixed linear systems

(Source file: 'skit/plib2/solver_abtb.h')

Synopsis

```

solver_abtb stokes      (a,b,mp);
solver_abtb elasticity (a,b,c,mp);

```

Description

The `solver_abtb` class provides direct or iterative algorithms for some mixed problem:

$$\begin{bmatrix} A & B^T \\ & & \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} Mf \\ Mg \end{bmatrix}$$

where A is symmetric positive definite and C is symmetric positive. By default, iterative algorithms are considered for tridimensional problems and direct methods otherwise. An optional argument can change this behavior. Such mixed linear problems appears for instance with the discretization of Stokes problems. The C matrix can be zero and then the corresponding argument can be omitted when invoking the constructor. Non-zero C matrix appears for of Stokes problems with stabilized P1-P1 element, or for nearly incompressible elasticity problems.

Direct algorithm

When the kernel of B^T is not reduced to zero, then the pressure p is defined up to a constant and the system is singular. In the case of iterative methods, this is not a problem.

But when using direct method, the system is then completed to impose a constraint on the pressure term and the whole matrix is factored one time for all.

Iterative algorithm

The preconditionned conjugate gradient algorithm is used, where the `mp` matrix is used as preconditionner. See see (undefined) [mixed_solver algorithm], page (undefined). The linear sub-systems related to the `A` matrix are also solved by an iterative algorithm. Use a second optional argument to change this default behavior: a factorization and a direct solver can be considered for these sub-systems.

Examples

See the user's manual for practical examples for the nearly incompressible elasticity, the Stokes and the Navier-Stokes problems.

Implementation

```
template <class T, class M = rheo_default_memory_model>
class solver_abtb_basic {
public:

    // typedefs:

    typedef typename csr<T,M>::size_type size_type;

    // allocators:

    solver_abtb_basic ();
    solver_abtb_basic (const csr<T,M>& a, const csr<T,M>& b, const csr<T,M>& mp,
        const solver_option& opt      = solver_option(),
        const solver_option& sub_opt = solver_option());
    solver_abtb_basic (const csr<T,M>& a, const csr<T,M>& b, const csr<T,M>& c, const
        const solver_option& opt      = solver_option(),
        const solver_option& sub_opt = solver_option());

    // accessors:

    void solve (const vec<T,M>& f, const vec<T,M>& g, vec<T,M>& u, vec<T,M>& p) const
    const solver_option& option() const { return _opt; }

protected:
    // internal
    void init();
    // data:
    mutable solver_option _opt;
    mutable solver_option _sub_opt;
    csr<T,M>              _a;
    csr<T,M>              _b;
```

```

    csr<T,M>          _c;
    csr<T,M>          _mp;
    solver_basic<T,M> _sA;
    solver_basic<T,M> _sa;
    solver_basic<T,M> _smp;
    bool              _need_constraint;
};
typedef solver_abtb_basic<Float,rheo_default_memory_model> solver_abtb;

```

4.31 solver_option - options for direct or iterative solvers

(Source file: 'skit/plib2/solver_option.h')

Description

This class implements a set of options for direct or iterative solvers, as implemented by the `[solver class]`, page `[undefined]`. An instance of this class can be supplied to the `solver` constructor.

Options

iterative

Indicate if we want to use an iterative solver or a direct one. Default is direct for 1D or 2D problems and iterative for 3D ones. Direct solver is `mumps` when available, otherwise `umfpack` or builtin solver when none of these solvers are available at Rheolef configure time. Iterative solver is `cg` for symmetric positive definite systems and `gmres` otherwise.

Options for iterative solvers

- tol** Tolerance for the stopping criterion. Default is the machine epsilon for the default `Float` type. The default `Float` type is `double` defined at Rheolef configure time and is `double` by default, when no special configure option is used.
- max_iter** Maximum number of iteration when using iterative method.
- absolute_stopping** Absolute or relative stopping criterion. With the absolute criterion, the algorithm stops when `norm(A*x-b) < tol`, otherwise it stops when `norm(A*x-b) < tol*norm(b)`. Default is to use an absolute stopping criterion.
- residue** On return, gives the obtained residue, optionally divided by the initial one when using a relative stopping. It is less or equal to `tol` when the iterative algorithm stops with success.
- n_iter** On return, gives the number of iterations performed. It is always less or equal to `max_iter` when the iterative algorithm stops with success.
- p_err** A pointer to the `odiststeam` standard error where residues are printed during iterations. When this pointer is zero, no errors are printed. Default is to print to `derr`.

label When printing errors, each line is prefixed by `[label]`. When the label is empty, each iterative algorithm uses its default label, e.g. `"cg"` for the conjugate gradient `cg`. By default the label is empty and this option is used to superset the algorithm default.

krylov_dimension
The dimension of the Krylov space used by the `gmres` algorithm. Default is `krylov_dimension=6`.

Options for direct solvers

n_refinement
Number of iterative refinement, when using direct method (`umfpack` only support it). This option is only active when using iterative solvers.

compute_determinant
Compute also the determinant of the matrix. This option is only active when using direct solvers. Requires `mumps` or `umfpack`.

preferred_library
When both `mumps` and `umfpack` are available, then `mumps` is the default. This option allows one to force to choose `umfpack`, by setting the string to `"umfpack"`. This option is only active when using direct solvers.

Options for developers

verbose_level
Can be set to 0, 1, 2 or 3. The default is 0.

level_of_fill
Built an incomplete factorization with the prescribed level of fill [1:5].

do_check Performs extra checks for debug.

force_seq
In distributed mode, restrict the linear system resolution to diagonal blocs per process. This option is only active when using the `mumps` direct solver.

ooc Out-of-core limit (Mo/percent depending on compilation options). In development.

amalgamation
Level of amalgamation [10:70] for Kass. In development.

Implementation

```
class solver_option {
public:
    typedef std::size_t size_type;
    static const long int decide = -1;
    mutable long int    iterative;
    Float              tol;
```

```

size_type      max_iter;
bool           absolute_stopping;
mutable Float  residue;
mutable size_type n_iter;
odiststream*   p_err;
mutable std::string label;
size_type      krylov_dimension;
size_type      n_refinement;
bool           compute_determinant;
std::string    preferred_library;
size_type      verbose_level;
bool           do_check;
bool           force_seq;
size_type      level_of_fill;
size_type      amalgamation;
size_type      ooc;

// allocator and default values:

solver_option()
: iterative      (decide),
#ifdef _RHEOLEF_HAVE_FLOAT128
  tol            (1e6*std::numeric_limits<Float>::epsilon()),
#else
  tol            (std::numeric_limits<Float>::epsilon()),
#endif
  max_iter       (100000),
  absolute_stopping (true),
  residue        (0),
  n_iter         (0),
  p_err          (&derr),
  label          (),
  krylov_dimension (6),
  n_refinement   (2),
  compute_determinant(false),
  preferred_library(),
  verbose_level  (0),
  do_check       (false),
  force_seq      (false),
  level_of_fill  (1),
  amalgamation   (10),
  ooc            (20000)
{
}

solver_option (const solver_option&);
solver_option& operator= (const solver_option&);
};

```

4.32 vec - vector in distributed environment

(Source file: 'skit/plib2/vec.h')

Synopsis

STL-like vector container for a sequential or distributed memory machine model. Additional operation from classical algebra.

Example

A sample usage of the class is:

```
int main(int argc, char**argv) {
    environment distributed(argc, argv);
    vec<double> x(100, 3.14);
    dout << x << endl;
}
```

Implementation note

Implementation use `disarray<T,M>`.

Implementation

```
template <class T, class M = rheo_default_memory_model>
class vec : public disarray<T, M> {
public:

    // typedef:

    typedef disarray<T, M> base;
    typedef T value_type;
    typedef typename base::size_type size_type;
    typedef std::ptrdiff_t difference_type;
    typedef basic_range<size_type, difference_type> range_type;
    typedef typename base::reference reference;
    typedef typename base::const_reference const_reference;
    typedef typename base::iterator iterator;
    typedef typename base::const_iterator const_iterator;
    typedef typename float_traits <value_type>::type float_type;

    // allocator/deallocator:

    vec (const vec<T,M>&);
    vec<T,M>& operator= (const vec<T,M>& x);

    vec (const distributor& ownership,
         const T& init_val = std::numeric_limits<T>::max());
```

```

    vec(size_type dis_size = 0,
        const T& init_val = std::numeric_limits<T>::max());

    void resize (
        const distributor& ownership,
        const T& init_val = std::numeric_limits<T>::max());

    void resize (
        size_type size = 0,
        const T& init_val = std::numeric_limits<T>::max());

// accessors:

    const_reference operator[] (size_type i) const;
    reference          operator[] (size_type i);

    T min () const;
    T max () const;
    T max_abs () const;

// range:

    vec(const vec_range<T,M>& vr);
    vec(const vec_range_const<T,M>& vr);
    vec<T,M>& operator= (const vec_range<T,M>& vr);
    vec<T,M>& operator= (const vec_range_const<T,M>& vr);

    vec_range_const<T,M> operator[] (const range_type& r) const;
    vec_range<T,M>          operator[] (const range_type& r);

// assignment to a constant:

    vec<T,M>& operator= (const int& expr);
    vec<T,M>& operator= (const T& expr);

// expression template:

    template <class Expr,
              class Sfinae
              = typename std::enable_if<
                  details::is_vec_expr_v2_arg<Expr>::value
                  && ! details::is_vec<Expr>::value
              >::type>
    vec (const Expr& expr);

    template <class Expr,
              class Sfinae
              = typename std::enable_if<

```



```

                                details::is_vec_expr_v2_arg<Expr>::value
                                && ! details::is_vec<Expr>::value
                                >::type>
    vec<T, M>& operator= (const Expr& expr);

    // initializer list (c++ 2011):

#ifdef _RHEOLEF_HAVE_STD_INITIALIZER_LIST
    vec (const std::initializer_list<vec_concat_value<T,M> >& init_list);
    vec<T,M>& operator= (const std::initializer_list<vec_concat_value<T,M> >& init_
#endif // _RHEOLEF_HAVE_STD_INITIALIZER_LIST
};

```

4.33 irheostream, orheostream - large data streams

(Source file: 'util/lib/rheostream.h')

Abstract

This class provides a stream interface for large data management. File decompression is assumed using `gzip` and a recursive search in a directory list is provided for input.

```
orheostream foo("NAME", "suffix");
```

is like

```
ofstream foo("NAME.suffix").
```

However, if *NAME* does not end with '`.suffix`', then '`.suffix`' is automatically added. By default, compression is performed on the fly with `gzip`, adding an additional '`.gz`' suffix. The `flush` action is nicely handled in compression mode:

```
foo.flush();
```

This feature allows intermediate results to be available during long computations. The compression can be deactivated while opening a file by an optional argument:

```
orheostream foo("NAME", "suffix", io::nogz);
```

An existing compressed file can be reopen in `append` mode: new results will be appended at the end of an existing file:

```
orheostream foo("NAME", "suffix", io::app);
```

Conversely,

```
irheostream foo("NAME","suffix");
```

is like

```
ifstream foo("NAME.suffix").
```

However, we look at a search path environment variable `RHEOPATH` in order to find *NAME* while suffix is assumed. Moreover, `gzip` compressed files, ending with the `‘.gz’` suffix is assumed, and decompression is done.

Finally, a set of useful functions are provided.

Description

The following code:

```
irheostream is("results", "data");
```

will recursively look for a `‘results[data.gz]’` file in the directory mentioned by the `RHEOPATH` environment variable.

For instance, if you insert in our `“.cshrc”` something like:

```
setenv RHEOPATH " ./home/dupont:/usr/local/math/demo"
```

the process will study the current directory `‘.’`, then, if neither `‘square.data.gz’` nor `‘square.data’` exists, it scan all subdirectory of the current directory. Then, if file is not founded, it start recusively in `‘/home/dupond’` and then in `‘/usr/local/math/demo’`.

File decompression is performed by using the `gzip` command, and data are pipe-lined directly in memory.

If the file start with `‘.’` as `‘./square’` or with a `‘/’` as `‘/home/oscar/square’`, no search occurs and `RHEOPATH` environment variable is not used.

Also, if the environment variable `RHEOPATH` is not set, the default value is the current directory `‘.’`.

For output stream:

```
orheostream os("newresults", "data");
```

file compression is assumed, and `"newresults.data.gz"` will be created.

File loading and storing are mentioned by a message, either:

```
! load "./results.data.gz"
```

or:

```
! file "./newresults.data.gz" created.
```

on the `clog` stream. By adding the following:

```
clog << noverbose;
```

you turn off these messages (see [\[iorheo ialgorithm\]](#), page [\[undefined\]](#)).

Implementation

```

class irheostream : public boost::iostreams::filtering_stream<boost::iostreams::input>
public:
    irheostream() : boost::iostreams::filtering_stream<boost::iostreams::input>() {}
    irheostream(const std::string& name, const std::string& suffix = std::string())
    virtual ~irheostream();
    void open (const std::string& name, const std::string& suffix = std::string())
    void close();
protected:
    std::ifstream _ifs;
};

static const bool dont_gzip = false;
class orheostream : public boost::iostreams::filtering_stream<boost::iostreams::output>
public:
    orheostream() : boost::iostreams::filtering_stream<boost::iostreams::output>() {}
    orheostream(const std::string& name, const std::string& suffix = std::string(),
        io::mode_type mode = io::out);
    virtual ~orheostream();
    void open (const std::string& name, const std::string& suffix = std::string(),
        io::mode_type mode = io::out);
    void flush();
    void close();
    const std::string& filename() const { return _full_name; }
protected:
    void _open_internal (io::mode_type mode);
    void _close_internal ();
// data:
    io::mode_type _mode;
    std::string _full_name;
};

std::string itos (std::string::size_type i);
std::string ftos (const Float& x);

// catch first occurrence of string in file
bool scatch (std::istream& in, const std::string& ch, bool full_match = true);

// has_suffix("toto.suffix", "suffix") -> true
bool has_suffix (const std::string& name, const std::string& suffix);

// "toto.suffix" --> "toto"
std::string delete_suffix (const std::string& name, const std::string& suffix);

// has_any_suffix("toto.any_suffix") -> true
bool has_any_suffix (const std::string& name);

// delete_any_suffix("toto.any_suffix") --> "toto"
std::string delete_any_suffix (const std::string& name);

```

```
// "/usr/local/dir/toto.suffix" --> "toto.suffix"
std::string get_basename (const std::string& name);

// "/usr/local/dir/toto.suffix" --> "/usr/local/dir"
std::string get_dirname (const std::string& name);

// "toto" --> "/usr/local/math/data/toto.suffix"
std::string get_full_name_from_rheo_path (const std::string& rootname, const std::s

// "." + "../geodir" --> "../../geodir"
void append_dir_to_rheo_path (const std::string& dir);

// "../geodir" + "." --> "../geodir:."
void prepend_dir_to_rheo_path (const std::string& dir);

bool file_exists (const std::string& filename);

// string to float
bool is_float (const std::string&);
Float to_float (const std::string&);

// in TMPDIR environment variable or "/tmp" by default
std::string get_tmpdir();
```


5 Algorithms

5.1 adapt - mesh adaptation

(Source file: 'nfem/plib/adapt.h')

Synopsis

```
geo adapt (const field& phi); geo adapt (const field& phi, const adapt_option& opts);
```

Description

The function `adapt` implements the mesh adaptation procedure, based on the `gmsh` (isotropic) or `bamg` (anisotropic) mesh generators. The `bamg` mesh generator is the default in two dimension. For dimension one or three, `gmsh` is the only generator supported yet. In the two dimensional case, the `gmsh` correspond to the `opts.generator="gmsh"`.

The strategy based on a metric determined from the Hessian of a scalar governing field, denoted as `phi`, and that is supplied by the user. Let us denote by `H=Hessian(phi)` the Hessian tensor of the field `phi`. Then, `|H|` denote the tensor that has the same eigenvector as `H`, but with absolute value of its eigenvalues:

$$|H| = Q * \text{diag}(|\lambda_i|) * Q^t$$

The metric `M` is determined from `|H|`. Recall that an isotropic metric is such that $M(x) = h_{loc}(x)^{-2} * Id$ where `hloc(x)` is the element size field and `Id` is the identity $d \times d$ matrix, and $d=1,2,3$ is the physical space dimension.

Gmsh isotropic metric

$$M(x) = \frac{\max_{(i=0 \dots d-1)} (|\lambda_i(x)|) * Id}{err * hcoef^2 * (\sup_y(\phi(y)) - \inf_y(\phi(y)))}$$

Notice that the denominator involves a global (absolute) normalization $\sup_y(\phi(y)) - \inf_y(\phi(y))$ of the governing field `phi` and the two parameters `opts.err`, the target error, and `opts.hcoef`, a secondary normalization parameter (defaults to 1).

Bamg anisotropic metric

There are two approach for the normalization of the metric. The first one involves a global (absolute) normalization:

$$M(x) = \frac{|H(x)|}{err * hcoef^2 * (\sup_y(\phi(y)) - \inf_y(\phi(y)))}$$

The first one involves a local (relative) normalization:

$$M(x) = \frac{|H(x)|}{err * hcoef^2 * (|\phi(x)|, cutoff * \max_y |\phi(y)|)}$$

Notice that the denominator involves a local value `phi(x)`. The parameter is provided by the optional variable `opts.cutoff`; its default value is `1e-7`. The default strategy is the local normalization. The global normalization can be enforced by setting `opts.additional="-AbsError"`.

When choosing global or local normalization ?

When the governing field `phi` is bounded, i.e. when `err*hcoef^2*(sup_y(phi(y))-inf_y(phi(y)))` will converge versus mesh refinement to a bounded value, the global normalization defines a metric that is mesh-independent and thus the adaptation loop will converge.

Otherwise, when `phi` presents singularities, with unbounded values (such as corner singularity, i.e. presents peaks when represented in elevation view), then the mesh adaptation procedure is more difficult. The global normalization divides by quantities that can be very large and the mesh adaptation can diverges when focusing on the singularities. In that case, the local normalization is preferable. Moreover, the focus on singularities can also be controlled by setting `opts.hmin` not too small.

The local normalization has been chosen as the default since it is more robust. When your field `phi` does not present singularities, then you can switch to the global numbering that leads to a best equirepartition of the error over the domain.

Implementation

```
struct adapt_option {
    typedef std::vector<int>::size_type size_type;
    std::string generator;
    bool isotropic;
    Float err;
    Float errg;
    Float hcoef;
    Float hmin;
    Float hmax;
    Float ratio;
    Float cutoff;
    size_type n_vertices_max;
    size_type n_smooth_metric;
    bool splitpbedge;
    Float thetaquad;
    Float anisomax;
    bool clean;
    std::string additional;
    bool double_precision;
    Float anglecorner; // angle below which bamg considers 2 consecutive edge to be
                      // the same spline
    adapt_option() :
        generator(""),
```

```

        isotropic(true), err(1e-2), errg(1e-1), hcoef(1), hmin(0.0001), hmax(0.3),
        n_vertices_max(50000), n_smooth_metric(1),
        splitpbedge(true), thetaquad(std::numeric_limits<Float>::max()),
        anisomax(1e6), clean(false), additional("-RelError"), double_precision(false),
        anglecorner(0)
    {}
};
template <class T, class M>
geo_basic<T,M>
adapt (
    const field_basic<T,M>& phi,
    const adapt_option& options = adapt_option());

```

5.2 compose - a n-ary function with n fields

(Source file: 'nfem/plib/compose.h')

Description

Compose a n-ary function f with n fields.

```

        geo omega ("circle");
        space Xh (omega, "P1");
        field uh (Xh, 1.0);
        field vh = interpolate (compose(f,uh));

```

The `compose` operator could be used in all non-linear expressions involved in either the `interpolate` or the `integrate` functions (see [\[interpolate algorithm\]](#), page [\[undefined\]](#) and [\[integrate algorithm\]](#), page [\[undefined\]](#)). The `f` function could be either a usual function or a functor.

Characteristic

The `compose` function supports also the characteristic algorithm (see [\[characteristic class\]](#), page [\[undefined\]](#)) used for convection.

```

        characteristic X (-delta_t*uh);
        test v (Xh);
        field lh = integrate (compose(uh,X)*v);

```

Implementation

The n-arity bases on the variadic template feature of the 2011 c++ normalisation. When this feature is not available, only unary and binary functions are supported.

5.3 continuation – continuation algorithm for nonlinear problems

(Source file: 'nfem/plib/continuation.h')

Description

This function applies a continuation algorithm for the resolution of the following problem:

$$F(\lambda, u) = 0$$

where `lambda` is a parameter and `u` is the corresponding solution. The main idea is to follow a branch of solution `u(lambda)` when the parameter `lambda` varies. A simple call to the algorithm writes:

```
my_problem P;
field uh (Vh,0);
continuation (P, uh, &dout, &derr);
```

The continuation algorithm bases on the `damped_newton` (see [\(undefined\)](#) [`damped_newton` algorithm], page [\(undefined\)](#)) and thus requirement for the `my_problem` class may contains methods for the evaluation of `F` (aka the residue) and its derivative:

```
class my_problem {
public:
    typedef value_type;
    typedef float_type;
    string parameter_name() const;
    float_type parameter() const;
    void set_parameter (float_type lambda);
    value_type residue      (const value_type& uh) const;
    void update_derivative  (const value_type& uh) const;
    csr<float_type> derivative (const value_type& uh) const;
    value_type derivative_versus_parameter (const field& uh) const;
    value_type derivative_solve      (const value_type& mrh) const;
    value_type derivative_trans_mult (const value_type& mrh) const;
    bool stop (const value_type& xh) const;
    idiststream& get (idiststream& is,      value_type& uh);
    odiststream& put (odiststream& os, const value_type& uh) const;
    float_type space_norm (const value_type& uh) const;
    float_type dual_space_norm (const value_type& mrh) const;
    float_type space_dot (const value_type& xh, const value_type& yh) const;
    float_type dual_space_dot (const value_type& mrh, const value_type& msh) const;
    value_type massify      (const value_type& uh) const;
    value_type unmassify    (const value_type& mrh) const;
};
```

See the example `combustion.h` in the user's documentation for more. The optional argument of type class `continuation_option` (see [\(undefined\)](#) [`continuation_option` class], page [\(undefined\)](#)) allows one to control some features of the algorithm.

Implementation

```
template<class Problem>
void
continuation (
```

```

Problem&                               F,
typename Problem::value_type&         uh,
odiststream*                           p_out,
odiststream*                           p_err,
const continuation_option& opts = continuation_option()

```

5.4 damped_newton – damped Newton nonlinear algorithm

(Source file: ‘nfem/plib/damped_newton.h’)

Description

Nonlinear damped Newton algorithm for the resolution of the following problem:

$$F(u) = 0$$

A simple call to the algorithm writes:

```

my_problem P;
field uh (Vh);
damped_newton (P, uh, tol, max_iter);

```

In addition to the members required for the Newton algorithm (see [\[newton algorithm\]](#), page [\[undefined\]](#)), the `space_norm` is required for the damped Newton line search algorithm:

```

class my_problem {
public:
    ...
    value_type derivative_trans_mult (const value_type& mrh) const;
    Float space_norm (const value_type& uh) const;
};

```

Implementation

```

template <class Problem, class Field, class Real, class Size>
int damped_newton (const Problem& F, Field& u, Real& tol, Size& max_iter, odiststre

```

5.5 integrate - integrate a function or an expression

(Source file: ‘nfem/plib/integrate.h’)

Description

Integrate an expression over a domain by using a quadrature formulae. There are three main usages of the integrate function, depending upon the type of the expression. (i) When the expression is a numerical one, it leads to a numerical value. (ii) When the expression involves a symbolic test-function see [\[test class\]](#), page [\[undefined\]](#), the result is a linear form, represented by the `field` class. (iii) When the expression involves both symbolic trial- and test-functions see [\[test class\]](#), page [\[undefined\]](#), the result is a bilinear form, represented by the `field` class.

Synopsis

```

Float integrate (geo domain);
Float integrate (geo domain, quadrature_option qopt);
Value integrate (geo domain, Expression, quadrature_option qopt);

field integrate (Expression);
field integrate (Expression, quadrature_option qopt);
field integrate (geo domain, Expression);
field integrate (geo domain, Expression, quadrature_option qopt);

form integrate (Expression);
form integrate (Expression, integrate_option qopt);
form integrate (geo domain, Expression);
form integrate (geo domain, Expression, integrate_option qopt);

```

Example

For computing the measure of a domain:

```
Float meas_omega = integrate (omega);
```

For computing the integral of a function:

```

Float f (const point& x);
...
quadrature_option qopt;
qopt.set_order (3);
Float int_f = integrate (omega, f, qopt);

```

The last argument specifies the quadrature formulae (see [\[quadrature_option class\]](#), page [\(undefined\)](#)) used for the computation of the integral. The function can be replaced by any field-valued expression (see [\[field class\]](#), page [\(undefined\)](#)). For computing a right-hand-side of a variational formulation with the previous function `f`:

```

space Xh (omega, "P1");
test v (Xh);
field lh = integrate (f*v);

```

For computing a bilinear form:

```

trial u (Xh);
test v (Xh);
form m = integrate (u*v);

```

The expression `u*v` can be replaced by any bilinear expression (see [\[field class\]](#), page [\(undefined\)](#)).

Default arguments

In the case of a linear or bilinear form, the domain is optional: by default it is the full domain definition of the test function. Also, the quadrature formulae is optional: by default, its order is $2*k+1$ where k is the polynomial degree of the `Xh` space associated to the test function `v`. When both a test `u` and trial `v` functions are supplied, let k_1 and k_2 be their polynomial degrees. Then the default quadrature is chosen to be exact at least for k_1+k_2+1

polynoms. When the integration is performed on a subdomain, this subdomain simply replace the first argument and a domain name could also be used:

```
field l2h = integrate (omega["boundary"], f*v);
field l3h = integrate ("boundary", f*v);
```

For convenience, only the domain name can be supplied.

5.6 interpolate - Lagrange interpolation of a function

(Source file: 'nfem/plib/interpolate.h')

Description

The function `interpolation` implements the Lagrange interpolation of a function or a class-function.

Synopsis

```
template <class Function> field interpolate (const space& Xh, const Function& f);
```

Example

The following code compute the Lagrange interpolation `pi_h_u` of `u(x)`.

```
Float u(const point& x);
...
geo omega("square");
space Xh (omega, "P1");
field pi_h_u = interpolate (Xh, u);
```

Advanced example

It is possible the replace the function `u` by a variable of the `field` type that represents a piecewise polynomial function: this invocation allows the reinterpolation of a field on another mesh or with another approximation.

```
geo omega2 ("square2");
space X2h (omega2, "P1");
field uh2 = interpolate (X2h, pi_h_u);
```

5.7 level_set - compute a level set from a function

(Source file: 'nfem/plib/level_set.h')

Synopsis

```
geo level_set (const field& fh);
```

Description

Given a function `fh` defined in a domain `Lambda`, compute the level set defined by $\{x \text{ in } \text{Lambda}, fh(x) = 0\}$. This level set is represented by the `geo` class.

Options

The option class `leve_set_option_type` controls the slit of quadrilaterals into triangles for tridimensional intersected surface and also the zero machine precision, `epsilon`.

Implementation

```
struct level_set_option {
    bool split_to_triangle;
    Float epsilon;
    level_set_option()
        : split_to_triangle(true),
          epsilon(100*std::numeric_limits<Float>::epsilon())
    {}
};
template <class T, class M>
geo_basic<T,M> level_set (
    const field_basic<T,M>& fh,
    const level_set_option& opt = level_set_option());
```

5.8 limiter - discontinuous Galerkin slope limiter

(Source file: 'nfem/plib/limiter.h')

Synopsis

field limiter (const field& uh, options...);

Description

This function returns a slope limited field for any supplied discontinuous approximation.

```
geo omega ("square");
space Xh (omega, "P1d");
field uh (Xh);
...
field vh = limiter(uh);
```

This function is still in development as a prototype: it supports only $d=1$ dimension and $k=0$ or 1 polynomial degrees. Its generalization to 2D and 3D geometries and any polynomial degree is in development.

Implementation

```
struct limiter_option_type {
    bool active;
    Float theta; // > 1, see Coc-1998, P. 209
    Float M;      // M=max|u''(t=0)(x)| at x where u'(t)(x)=0 :extremas
    limiter_option_type() : active(true), theta(1.5), M(1) {}
};
```

```

template <class T, class M>
field_basic<T,M>
limiter (
    const field_basic<T,M>& uh,
    const T& bar_g_S = 1.0,
    const limiter_option_type& opt = limiter_option_type());

```

5.9 newton – Newton nonlinear algorithm

(Source file: ‘nfem/plib/newton.h’)

Description

Nonlinear Newton algorithm for the resolution of the following problem:

$$F(u) = 0$$

A simple call to the algorithm writes:

```

my_problem P;
field uh (Vh);
newton (P, uh, tol, max_iter);

```

The `my_problem` class may contains methods for the evaluation of F (aka residue) and its derivative:

```

class my_problem {
public:
    typedef value_type;
    value_type residue (const value_type& uh) const;
    Float dual_space_norm (const value_type& mrh) const;
    void update_derivative (const value_type& uh) const;
    value_type derivative_solve (const value_type& mrh) const;
};

```

The `dual_space_norm` returns a scalar from the weighted residual field term `mrh` returned by the `residue` function: this scalar is used as stopping criteria for the algorithm. The `update_derivative` and `derivative_solver` members are called at each step of the Newton algorithm. See the example `p_laplacian.h` in the user’s documentation for more.

Implementation

```

template <class Problem, class Field>
int newton (const Problem& P, Field& uh, Float& tol, size_t& max_iter, ostream& p_derr) {
    if (p_derr) *p_derr << "# Newton:" << std::endl << "# n r" << std::endl << std::flush;
    for (size_t n = 0; true; n++) {
        Field rh = P.residue(uh);
        Float r = P.dual_space_norm(rh);
        if (p_derr) *p_derr << n << " " << r << std::endl << std::flush;
        if (r <= tol) { tol = r; max_iter = n; return 0; }
    }
}

```

```

        if (n == max_iter) { tol = r; return 1; }
        P.update_derivative (uh);
        Field delta_uh = P.derivative_solve (-rh);
        uh += delta_uh;
    }
}

```

5.10 riesz - approximate a Riesz representer

(Source file: 'nfem/plib/riesz.h')

Synopsis

The `riesz` function is now obsolete: it has been now superseded by the `integrate` function see [\[integrate algorithm\]](#), page [\[undefined\]](#).

```

template <class Expr>
field riesz (space, Expr expr);
field riesz (space, Expr expr, quadrature_option);
field riesz (space, Expr expr, domain);
field riesz (space, Expr expr, domain, quadrature_option);

```

The domain can be also provided by its name as a string. The old-fashioned code:

Note

The `riesz` function is now obsolete: it has been now superseded by the `integrate` function see [\[integrate algorithm\]](#), page [\[undefined\]](#). The old-fashioned code:

```

field l1h = riesz (Xh, f);
field l2h = riesz (Xh, f, "boundary");

```

writes now:

```

test v (Xh);
field l1h = integrate (f*v);
field l2h = integrate ("boundary", f*v);

```

The `riesz` function is still present in the library for backward compatibility purpose.

Description

Let f be any continuous function, its Riesz representer in the finite element space X_h on the domain Ω is defined by:

$$\text{dual}(l_h, v_h) = \frac{\int_{\Omega} f(x) v_h(x) dx}{\int_{\Omega} v_h(x) dx}$$

for all v_h in X_h , where `dual` denotes the duality between X_h and its dual. As X_h is a finite dimensional space, its dual is identified as X_h and the duality product as the Euclidian one. The Riesz representer is thus the l_h field of X_h where its i -th degree of freedom is:

$$\text{dual}(\text{lh}, \text{vh}) = \frac{\int_{\Omega} f(x) \phi_i(x) dx}{\int_{\Omega} \phi_i(x) dx}$$

where ϕ_i is the i -th basis function in X_h . The integral is evaluated by using a quadrature formula. By default the quadrature formula is the Gauss one with the order equal to $2k-1$ where k is the polynomial degree in X_h . Alternative quadrature formula and order is available by passing an optional variable to `riesz`.

The function `riesz` implements the approximation of the Riesz representer by using some quadrature formula for the evaluation of the integrals. Its argument can be any function, class-function or linear or nonlinear expressions mixing fields and continuous functions.

Example

The following code compute the Riesz representant, denoted by `lh` of $f(x)$, and the integral of f over the domain `omega`:

```
Float f(const point& x);
...
space Xh (omega_h, "P1");
field lh = riesz (Xh, f);
Float int_f = dual(lh, 1);
```

Options

An optional argument specifies the quadrature formula used for the computation of the integral. The domain of integration is by default the mesh associated to the finite element space. An alternative domain `dom`, e.g. a part of the boundary can be supplied as an extra argument. This domain can be also a `band` associated to the banded level set method.

Implementation

```
template <class T, class M, class Function>
inline
field_basic<T,M>
riesz (
    const space_basic<T,M>&      Xh,
    const Function&              f,
    const quadrature_option& qopt
    = quadrature_option())
```

Implementation

```
template <class T, class M, class Function>
field_basic<T,M>
riesz (
    const space_basic<T,M>&      Xh,
    const Function&              f,
```



```

const geo_basic<T,M>&      dom,
const quadrature_option& qopt
    = quadrature_option())

```

Implementation

```

template <class T, class M, class Function>
field_basic<T,M>
riesz (
    const space_basic<T,M>&      Xh,
    const Function&              f,
    std::string                  dom_name,
    const quadrature_option& qopt
        = quadrature_option())

```

Implementation

```

template <class T, class M, class Function>
field_basic<T,M>
riesz (
    const space_basic<T,M>&      Xh,
    const Function&              f,
    const band_basic<T,M>&      gh,
    const quadrature_option& qopt
        = quadrature_option())

```

5.11 cg – conjugate gradient algorithm.

(Source file: 'skit/plib2/cg.h')

Synopsis

```

template <class Matrix, class Vector, class Preconditioner, class Real>
int cg (const Matrix &A, Vector &x, const Vector &b,
    const solver_option& sopt = solver_option())

```

Example

The simplest call to `cg` has the folling form:

```

solver_option sopt;
sopt.max_iter = 100;
sopt.tol = 1e-7;
int status = cg(a, x, b, eye(), sopt);

```

Description

`cg` solves the symmetric positive definite linear system $Ax=b$ using the conjugate gradient method. The return value indicates convergence within `max_iter` (input) iterations (0), or

no convergence within `max_iter` iterations (1). Upon successful return, output arguments have the following values:

`x` approximate solution to $Ax = b$

`sopt.n_iter`
 the number of iterations performed before the tolerance was reached

`sopt.residue`
 the residual after the final iteration

See also the `<undefined>` [solver_option class], page `<undefined>`.

Note

`cg` is an iterative template routine.

`cg` follows the algorithm described on p. 15 in

Templates for the solution of linear systems: building blocks for iterative methods, 2nd Edition, R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst, SIAM, 1994, ftp.netlib.org/templates/templates.ps.

The present implementation is inspired from IML++ 1.2 iterative method library, <http://math.nist.gov/impl/>

Implementation

```
template <class Matrix, class Vector, class Vector2, class Preconditioner>
int cg (const Matrix &A, Vector &x, const Vector2 &Mb, const Preconditioner &M,
        const solver_option& sopt = solver_option())
{
    typedef typename Vector::size_type   Size;
    typedef typename Vector::float_type  Real;
    std::string label = (sopt.label != "" ? sopt.label : "cg");
    Vector b = M.solve(Mb);
    Real norm2_b = dot(Mb,b);
    if (sopt.absolute_stopping || norm2_b == Real(0)) norm2_b = 1;
    Vector Mr = Mb - A*x;
    Real norm2_r = 0;
    if (sopt.p_err) (*sopt.p_err) << "[" << label << "]" #iteration residue" << std::endl;
    Vector p;
    for (sopt.n_iter = 0; sopt.n_iter <= sopt.max_iter; sopt.n_iter++) {
        Vector r = M.solve(Mr);
        Real prev_norm2_r = norm2_r;
        norm2_r = dot(Mr, r);
        sopt.residue = sqrt(norm2_r/norm2_b);
        if (sopt.p_err) (*sopt.p_err) << "[" << label << "]" " << sopt.n_iter << " " << sopt.residue << std::endl;
        if (sopt.residue <= sopt.tol) return 0;
        if (sopt.n_iter == 0) {
            p = r;
        } else {
            Real beta = norm2_r/prev_norm2_r;
            p = r + beta*p;
        }
    }
}
```

```

    }
    Vector Mq = A*p;
    Real alpha = norm2_r/dot(Mq, p);
    x += alpha*p;
    Mr -= alpha*Mq;
  }
  return 1;
}

```

5.12 diag - get diagonal part of a matrix

(Source file: 'skit/plib2/diag.h')

Description

This function get the diagonal part of a matrix.

```

csr<Float> a;
dia<Float> d = diag(a);

```

Todo

Build a csr matrix from a diagonal one or from a vector:

```

dia<Float> d;
csr<Float> a = diag(d);
vec<Float> u;
csr<Float> b = diag(u);

```

Implementation

```

template<class T, class M>
dia<T,M> diag (const csr<T,M>& a)

```

5.13 gmres – generalized minimum residual method

(Source file: 'skit/plib2/gmres.h')

Synopsis

```

template <class Matrix, class Vector, class Preconditioner,
          class SmallMatrix, class SmallVector, class Real, class Size>
int gmres (const Matrix &A, Vector &x, const Vector &b, const Preconditioner &M,
          SmallMatrix &H, const SmallVector& dummy,
          const solver_option& sopt);

```

Example

The simplest call to `gmres` has the following form:

```
solver_option sopt;
sopt.tol = 1e-7;
sopt.max_iter = 100;
size_t m = sopt.krylov_dimension;
boost::numeric::ublas::matrix<T> H(m+1,m+1);
vec<T,sequential> dummy(m,0.0);
int status = gmres (a, x, b, ic0(a), H, dummy, sopt);
```

Description

`gmres` solves the unsymmetric linear system $Ax = b$ using the generalized minimum residual method.

The return value indicates convergence within `max_iter` (input) iterations (0), or no convergence within `max_iter` iterations (1). Upon successful return, output arguments have the following values:

`x` approximate solution to $Ax = b$

`sopt.n_iter`
 the number of iterations performed before the tolerance was reached

`sopt.residue`
 the residual after the final iteration

See also the `[solver_option class]`, page `[undefined]`. In addition, `M` specifies a preconditioner, `H` specifies a matrix to hold the coefficients of the upper Hessenberg matrix constructed by the `gmres` iterations, `m` specifies the number of iterations for each restart.

`gmres` requires two matrices as input, `A` and `H`. The matrix `A`, which will typically be a sparse matrix) corresponds to the matrix in the linear system $Ax=b$. The matrix `H`, which will be typically a dense matrix, corresponds to the upper Hessenberg matrix `H` that is constructed during the `gmres` iterations. Within `gmres`, `H` is used in a different way than `A`, so its class must supply different functionality. That is, `A` is only accessed through its matrix-vector and transpose-matrix-vector multiplication functions. On the other hand, `gmres` solves a dense upper triangular linear system of equations on `H`. Therefore, the class to which `H` belongs must provide `H(i,j)` operator for element access.

Note

It is important to remember that we use the convention that indices are 0-based. That is `H(0,0)` is the first component of the matrix `H`. Also, the type of the matrix must be compatible with the type of single vector entry. That is, operations such as `H(i,j)*x(j)` must be able to be carried out.

`gmres` is an iterative template routine.

`gmres` follows the algorithm described on p. 20 in *Templates for the solution of linear systems: building blocks for iterative methods*, 2nd Edition, R. Barrett, M. Berry, T. F.

Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst, SIAM, 1994, <ftp.netlib.org/templates/templates.ps>.

The present implementation is inspired from IML++ 1.2 iterative method library, <http://math.nist.gov/iml>

Implementation

```

namespace details {
template <class SmallMatrix, class SmallVector, class Vector, class Vector2, class ...>
void update (Vector& x, Size k, const SmallMatrix& h, const SmallVector& s, Vector2& y) {
    SmallVector y = s;
    // back solve:
    for (int i = k; i >= 0; i--) {
        y(i) /= h(i,i);
        for (int j = i - 1; j >= 0; j--)
            y(j) -= h(j,i) * y(i);
    }
    for (Size j = 0; j <= k; j++) {
        x += v[j] * y(j);
    }
}

template<class Real>
void generate_plane_rotation (const Real& dx, const Real& dy, Real& cs, Real& sn) {
    if (dy == Real(0)) {
        cs = 1.0;
        sn = 0.0;
    } else if (abs(dy) > abs(dx)) {
        Real temp = dx / dy;
        sn = 1.0 / sqrt( 1.0 + temp*temp );
        cs = temp * sn;
    } else {
        Real temp = dy / dx;
        cs = 1.0 / sqrt( 1.0 + temp*temp );
        sn = temp * cs;
    }
}

template<class Real>
void apply_plane_rotation (Real& dx, Real& dy, const Real& cs, const Real& sn) {
    Real temp = cs * dx + sn * dy;
    dy = -sn * dx + cs * dy;
    dx = temp;
}
} // namespace details

template <class Matrix, class Vector, class Preconditioner,
          class SmallMatrix, class SmallVector>

int
gmres (const Matrix &A, Vector &x, const Vector &b, const Preconditioner &M,
       SmallMatrix &H, const SmallVector&, const solver_option& sopt = solver_option{0})
{
    typedef typename Vector::size_type Size;

```

```

typedef typename Vector::float_type Real;
std::string label = (sopt.label != "" ? sopt.label : "gmres");
Size m = sopt.krylov_dimension;
Vector w;
SmallVector s(m+1), cs(m+1), sn(m+1);
Real residue;
Real norm_b = norm(M.solve(b));
Vector r = M.solve(b - A * x);
Real beta = norm(r);
if (sopt.p_err) (*sopt.p_err) << "[" << label << "]" # norm_b=" << norm_b << std::e
                << "[" << label << "]" #iteration residue" << std::e
if (sopt.absolute_stopping || norm_b == Real(0)) norm_b = 1;
sopt.n_iter = 0;
sopt.residue = norm(r)/norm_b;
if (sopt.residue <= sopt.tol) return 0;
std::vector<Vector> v (m+1);
for (sopt.n_iter = 1; sopt.n_iter <= sopt.max_iter; ) {
    v[0] = r * (1.0 / beta);    // ??? r / beta
    s = 0.0;
    s(0) = beta;
    for (Size i = 0; i < m && sopt.n_iter <= sopt.max_iter; i++, sopt.n_iter++) {
        w = M.solve(A * v[i]);
        for (Size k = 0; k <= i; k++) {
            H(k, i) = dot(w, v[k]);
            w -= H(k, i) * v[k];
        }
        H(i+1, i) = norm(w);
        v[i+1] = w * (1.0 / H(i+1, i)); // ??? w / H(i+1, i)
        for (Size k = 0; k < i; k++) {
            details::apply_plane_rotation (H(k,i), H(k+1,i), cs(k), sn(k));
        }
        details::generate_plane_rotation (H(i,i), H(i+1,i), cs(i), sn(i));
        details::apply_plane_rotation (H(i,i), H(i+1,i), cs(i), sn(i));
        details::apply_plane_rotation (s(i), s(i+1), cs(i), sn(i));
        sopt.residue = abs(s(i+1))/norm_b;
        if (sopt.p_err) (*sopt.p_err) << "[" << label << "]" " << sopt.n_iter << " " <
        if (sopt.residue <= sopt.tol) {
            details::update (x, i, H, s, v);
            return 0;
        }
    }
    details::update (x, m - 1, H, s, v);
    r = M.solve(b - A * x);
    beta = norm(r);
    sopt.residue = beta/norm_b;
    if (sopt.p_err) (*sopt.p_err) << "[" << label << "]" " << sopt.n_iter << " " <<
    if (sopt.residue < sopt.tol) return 0;
}
return 1;

```

```
}
```

5.14 minres – conjugate gradient algorithm.

(Source file: 'skit/plib2/minres.h')

Synopsis

```
template <class Matrix, class Vector, class Preconditioner, class Real>
int minres (const Matrix &A, Vector &x, const Vector &b, const Preconditioner &M,
            const solver_option& sopt = solver_option);
```

Example

The simplest call to `minres` has the following form:

```
solver_option sopt;
sopt.max_iter = 100;
sopt.tol = 1e-7;
int status = minres (a, x, b, eye(), sopt);
```

Description

`minres` solves the symmetric positive definite linear system $Ax=b$ using the Conjugate Gradient method.

The return value indicates convergence within `max_iter` (input) iterations (0), or no convergence within `max_iter` iterations (1). Upon successful return, output arguments have the following values:

`x` approximate solution to $Ax = b$

`sopt.n_iter`
 the number of iterations performed before the tolerance was reached

`sopt.residue`
 the residual after the final iteration

Note

`minres` follows the algorithm described in "Solution of sparse indefinite systems of linear equations", C. C. Paige and M. A. Saunders, SIAM J. Numer. Anal., 12(4), 1975. For more, see <http://www.stanford.edu/group/SOL/software.html> and also the PhD "Iterative methods for singular linear equations and least-squares problems", S.-C. T. Choi, Stanford University, 2006, <http://www.stanford.edu/group/SOL/dissertations/sou-cheng-choi-thesis.pdf> at page 60. The present implementation style is inspired from IML++ 1.2 iterative method library, <http://math.nist.gov/impl++>.

Implementation

```

template <class Matrix, class Vector, class Preconditioner>
int minres (const Matrix &A, Vector &x, const Vector &Mb, const Preconditioner &M,
  const solver_option& sopt = solver_option())
{
  // Size &max_iter, Real &tol, odiststream *p_derr = 0
  typedef typename Vector::size_type Size;
  typedef typename Vector::float_type Real;
  std::string label = (sopt.label != "" ? sopt.label : "minres");
  Vector b = M.solve(Mb);
  Real norm_b = sqrt(fabs(dot(Mb,b)));
  if (sopt.absolute_stopping || norm_b == Real(0.)) norm_b = 1;
  Vector Mr = Mb - A*x;
  Vector z = M.solve(Mr);
  Real beta2 = dot(Mr, z);
  Real norm_r = sqrt(fabs(beta2));
  sopt.residue = norm_r/norm_b;
  if (sopt.p_err) (*sopt.p_err) << "[" << label << "]" #iteration residue" << std::e
    << "[" << label << "]" 0 " << sopt.residue << std::e
  if (beta2 < 0 || sopt.residue <= sopt.tol) {
    dis_warning_macro ("beta2 = " << beta2 << " < 0: stop");
    return 0;
  }
  Real beta = sqrt(beta2);
  Real eta = beta;
  Vector Mv = Mr/beta;
  Vector u = z/beta;
  Real c_old = 1.;
  Real s_old = 0.;
  Real c = 1.;
  Real s = 0.;
  Vector u_old (x.ownership(), 0.);
  Vector Mv_old (x.ownership(), 0.);
  Vector w (x.ownership(), 0.);
  Vector w_old (x.ownership(), 0.);
  Vector w_old2 (x.ownership(), 0.);
  for (sopt.n_iter = 1; sopt.n_iter <= sopt.max_iter; sopt.n_iter++) {
    // Lanczos
    Mr = A*u;
    z = M.solve(Mr);
    Real alpha = dot(Mr, u);
    Mr = Mr - alpha*Mv - beta*Mv_old;
    z = z - alpha*u - beta*u_old;
    beta2 = dot(Mr, z);
    if (beta2 < 0) {
      dis_warning_macro ("minres: machine precision problem");
      sopt.residue = norm_r/norm_b;
      return 2;
    }
  }
}

```



```

    }
    Real beta_old = beta;
    beta = sqrt(beta2);
    // QR factorisation
    Real c_old2 = c_old;
    Real s_old2 = s_old;
    c_old = c;
    s_old = s;
    Real rho0 = c_old*alpha - c_old2*s_old*beta_old;
    Real rho2 = s_old*alpha + c_old2*c_old*beta_old;
    Real rho1 = sqrt(sqr(rho0) + sqr(beta));
    Real rho3 = s_old2 * beta_old;
    // Givens rotation
    c = rho0 / rho1;
    s = beta / rho1;
    // update
    w_old2 = w_old;
    w_old = w;
    w = (u - rho2*w_old - rho3*w_old2)/rho1;
    x += c*eta*w;
    eta = -s*eta;
    Mv_old = Mv;
    u_old = u;
    Mv = Mr/beta;
    u = z/beta;
    // check residue
    norm_r *= s;
    sopt.residue = norm_r/norm_b;
    if (sopt.p_err) (*sopt.p_err) << "[" << label << "]" " << sopt.n_iter << " " <<
    if (sopt.residue <= sopt.tol) return 0;
}
return 1;
}

```

5.15 cg_abtb, cg_abtbc, minres_abtb, minres_abtbc – solvers for mixed linear problems

(Source file: 'skit/plib2/mixed_solver.h')

Synopsis

```

template <class Matrix, class Vector, class Solver, class Preconditioner>
int cg_abtb (const Matrix& A, const Matrix& B, Vector& u, Vector& p,
             const Vector& Mf, const Vector& Mg, const Preconditioner& S1,
             const Solver& inner_solver_A, const solver_option& sopt = solver_option());

template <class Matrix, class Vector, class Solver, class Preconditioner>
int cg_abtbc (const Matrix& A, const Matrix& B, const Matrix& C, Vector& u, Vector& p,
              const Vector& Mf, const Vector& Mg, const Preconditioner& S1,
              const Solver& inner_solver_A, const Solver& inner_solver_C, const solver_option& sopt = solver_option());

```

```
const Solver& inner_solver_A, const solver_option& sopt = solver_option());
```

The synopsis is the same with the minres algorithm.

Examples

See the user's manual for practical examples for the nearly incompressible elasticity, the Stokes and the Navier-Stokes problems.

Description

Preconditioned conjugate gradient algorithm on the pressure p applied to the stabilized stokes problem:

$$\begin{bmatrix} A & B^T \\ B & -C \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} Mf \\ Mg \end{bmatrix}$$

where A is symmetric positive definite and C is symmetric positive and semi-definite. Such mixed linear problems appears for instance with the discretization of Stokes problems with stabilized P1-P1 element, or with nearly incompressible elasticity. Formally $u = \text{inv}(A) * (Mf - B^T * p)$ and the reduced system writes for all non-singular matrix $S1$:

$$\text{inv}(S1) * (B * \text{inv}(A) * B^T) * p = \text{inv}(S1) * (B * \text{inv}(A) * Mf - Mg)$$

Uzawa or conjugate gradient algorithms are considered on the reduced problem. Here, $S1$ is some preconditioner for the Schur complement $S = B * \text{inv}(A) * B^T$. Both direct or iterative solvers for $S1 * q = t$ are supported. Application of $\text{inv}(A)$ is performed via a call to a solver for systems such as $A * v = b$. This last system may be solved either by direct or iterative algorithms, thus, a general matrix solver class is submitted to the algorithm. For most applications, such as the Stokes problem, the mass matrix for the p variable is a good $S1$ preconditioner for the Schur complement. The stopping criteria is expressed using the $S1$ matrix, i.e. in L2 norm when this choice is considered. It is scaled by the L2 norm of the right-hand side of the reduced system, also in $S1$ norm.

5.16 uzawa — Uzawa algorithm.

(Source file: 'skit/plib2/uzawa.h')

Synopsis

```
template <class Matrix, class Vector, class Preconditioner, class Real>
int uzawa (const Matrix &A, Vector &x, const Vector &b, const Preconditioner &M,
           const solver_option& sopt)
```

Example

The simplest call to 'uzawa' has the folling form:

```

solver_option sopt;
sopt.max_iter = 100;
sopt.tol = 1e-7;
int status = uzawa(A, x, b, eye(), sopt);

```

Description

uzawa solves the linear system $Ax=b$ using the Uzawa method. The Uzawa method is a descent method in the direction opposite to the gradient, with a constant step length 'rho'. The convergence is assured when the step length 'rho' is small enough. If matrix A is symmetric positive definite, please uses 'cg' that computes automatically the optimal descent step length at each iteration. The return value indicates convergence within max_iter (input) iterations (0), or no convergence within max_iter iterations (1). Upon successful return, output arguments have the following values:

x approximate solution to $Ax = b$

sopt.n_iter
 the number of iterations performed before the tolerance was reached

sopt.residue
 the residual after the final iteration

See also the [\[solver_option class\]](#), page [\[undefined\]](#).

Implementation

```

template<class Matrix, class Vector, class Preconditioner, class Real2>
int uzawa (const Matrix &A, Vector &x, const Vector &Mb, const Preconditioner &M,
           const Real2& rho, const solver_option& sopt = solver_option())
{
    typedef typename Vector::size_type Size;
    typedef typename Vector::float_type Real;
    std::string label = (sopt.label != "" ? sopt.label : "uzawa");
    Vector b = M.solve(Mb);
    Real norm2_b = dot(Mb,b);
    Real norm2_r = norm2_b;
    if (sopt.absolute_stopping || norm2_b == Real(0)) norm2_b = 1;
    if (sopt.p_err) (*sopt.p_err) << "[" << label << "]" #iteration residue" << std::e
    for (sopt.n_iter = 0; sopt.n_iter <= sopt.max_iter; sopt.n_iter++) {
        Vector Mr = A*x - Mb;
        Vector r = M.solve(Mr);
        norm2_r = dot(Mr, r);
        sopt.residue = sqrt(norm2_r/norm2_b);
        if (sopt.p_err) (*sopt.p_err) << "[" << label << "]" " << sopt.n_iter << " " <<
        if (sopt.residue <= sopt.tol) return 0;
        x -= rho*r;
    }
    return 1;
}

```

5.17 catchmark - ostream manipulator

(Source file: 'util/lib/catchmark.h')

Description

The `catchmark` is used for building labels used for input-output of vector-valued fields (see [\[field class\]](#), page [\[undefined\]](#)):

```
cin  >> catchmark("f")  >> fh;
cout << catchmark("u")  << uh
     << catchmark("w")  << wh
     << catchmark("psi") << psih;
```

Assuming its value for output is "u", the corresponding labels will be "#u0", "#u1", "#u2", ...

Implementation

```
class catchmark {
public:
    catchmark(const std::string& x);
    const std::string& mark() const { return _mark; }
    friend std::istream& operator >> (std::istream& is, const catchmark& m);
    friend std::ostream& operator << (std::ostream& os, const catchmark& m);
protected:
    std::string _mark;
};
```


6 Internal classes

6.1 domain_indirect - a named part of a finite element mesh

(Source file: 'nfem/plib/domain_indirect.h')

Description

The `domain_indirect` class defines a container for a part of a finite element mesh. This describes the connectivity of edges or faces. This class is useful for boundary condition setting.

Implementation note

The `domain` class is split into two parts. The first one is the `domain_indirect` class, that contains the main renumbering features: it acts as a indirect on a `geo` class (see [\[geo class\]](#), page [\[undefined\]](#)). The second one is the `domain` class, that simply contains two smart_pointers: one on a `domain_indirect` and the second on the `geo` where renumbering is acting. Thus, the `domain` class develops a complete `geo`-like interface, via the `geo_abstract_rep` pure virtual class derivation, and can be used by the `space` class (see [\[space class\]](#), page [\[undefined\]](#)). The split between `domain_indirect` and `domain` is necessary, because the `geo` class contains a list of `domain_indirect`. It cannot contains a list of `domain` classes, that refers to the `geo` class itself: a loop in reference counting leads to a blocking situation in the automatic deallocation.

Implementation

```
template <>
class domain_indirect_basic<sequential> : public smart_pointer<domain_indirect_rep<
public:

// typedefs:

    typedef domain_indirect_rep<sequential> rep;
    typedef smart_pointer<rep>                base;
    typedef rep::size_type                    size_type;
    typedef rep::iterator_ioige               iterator_ioige;
    typedef rep::const_iterator_ioige         const_iterator_ioige;

// allocators:

    domain_indirect_basic ();

    template <class T>
    domain_indirect_basic (
        const geo_basic<T,sequential>&    omega,
        const std::string&                 name,
        size_type                          map_dim,
```

```

        const communicator&      comm,
        const std::vector<size_type>& ie_list);

template <class U>
domain_indirect_basic (
    disarray<geo_element_auto<heap_allocator<size_type> >,sequential, heap_allocator<size_type>>
        d_tmp,
    const geo_basic<U, sequential>& omega,
    std::vector<index_set>*      ball);

void resize (size_type n);

// accessors:

size_type size()      const;
size_type dis_size() const;
const distributor& ownership() const;

const_iterator_ioige ioige_begin() const;
const_iterator_ioige ioige_end()   const;
iterator_ioige ioige_begin();
iterator_ioige ioige_end();

const geo_element_indirect& oige (size_type ioige) const;

void set_name (std::string name);
void set_map_dimension (size_type map_dim);
std::string name ()      const;
size_type map_dimension () const;

// i/o:

odistream& put (odistream&) const;
template <class T>
idistream& get (idistream& ips, const geo_rep<T,sequential>& omega, std::vector<index_set>&
};

```

Implementation

```

template <>
class domain_indirect_basic<distributed> : public smart_pointer<domain_indirect_rep<distributed>>
public:

// typedefs:

typedef domain_indirect_rep<distributed> rep;
typedef smart_pointer<rep>               base;
typedef rep::size_type                   size_type;
typedef rep::iterator_ioige              iterator_ioige;

```

```

        typedef rep::const_iterator_ioige      const_iterator_ioige;

// allocators:

    domain_indirect_basic ();
    template<class T>
    domain_indirect_basic (
        const geo_basic<T,distributed>& omega,
        const std::string&          name,
        size_type                    map_dim,
        const communicator&         comm,
        const std::vector<size_type>& ie_list);

// accessors/modifiers:

    size_type size()      const;
    size_type dis_size() const;
    const distributor& ownership() const;

    const geo_element_indirect& oige (size_type ioige) const;

    void set_name (std::string name);
    void set_map_dimension (size_type map_dim);
    std::string name () const;
    size_type map_dimension () const;

// distributed specific accessors:

    const_iterator_ioige ioige_begin() const;
    const_iterator_ioige ioige_end()   const;
        iterator_ioige ioige_begin();
        iterator_ioige ioige_end();

    const distributor& ini_ownership() const;
    size_type ioige2ini_dis_ioige (size_type ioige) const;
    size_type ini_ioige2dis_ioige (size_type ini_ioige) const;

// i/o:

    template <class T>
    idiststream& get (idiststream& ips, const geo_rep<T,distributed>& omega);
    template <class T>
    odiststream& put (odiststream& ops, const geo_rep<T,distributed>& omega) const;
};

```

6.2 geo_domain - a named part of a finite element mesh that behaves as a mesh

(Source file: 'nfem/plib/geo_domain.h')

Description

The `geo_domain` class defines a container for a part of a finite element mesh. This class re-describes the vertices, edges or faces in a compact way, i.e. by skipping unused elements from the surrounding mesh.

Implementation note

The `geo_domain` class conserves the link to the original mesh such that fields defined on a `geo_domain` can inter-operate with fields defined on the surrounding mesh.

6.3 `geo_domain_indirect_rep` - a named part of a finite element mesh

(Source file: 'nfem/plib/geo_domain_indirect.h')

Description

The `geo_domain_indirect_rep` class defines a container for a part of a finite element mesh. This describes the connectivity of edges or faces. This class is useful for boundary condition setting.

Implementation note

The `geo_domain_indirect_rep` class is split into two parts. The first one is the `domain_indirect` class, that contains the main renumbering features: it acts as an indirection on a `geo` class (see `<undefined>` [geo class], page `<undefined>`). The second one is the `geo` class itself, named here the background `geo`. Thus, the `geo_domain_indirect` class develops a complete `geo`-like interface, via the `geo_abstract_rep` pure virtual class derivation, and can be used by the `space` class (see `<undefined>` [space class], page `<undefined>`).

The split between `domain_indirect` and `geo_domain_indirect` is necessary, because the `geo` class contains a list of `domain_indirect`. The `geo` class cannot contain a list of `geo_domain_indirect` classes, that refers to the `geo` class itself: a loop in reference counting leads to a blocking situation in the automatic deallocation.

6.4 numbering - global degree of freedom numbering

(Source file: 'nfem/plib/numbering.h')

Synopsis

The `numbering` class defines methods that furnish global numbering of degrees of freedom. This numbering depends upon the degrees of polynoms on elements and upon the continuity requirement at inter-element boundary. For instance the "P1" continuous finite element approximation has one degree of freedom per vertice of the mesh, while its discontinuous counterpart has `dim(basis)` times the number of elements of the mesh, where `dim(basis)` is the size of the local finite element basis.

Implementation

```

template <class T, class M = rheo_default_memory_model>
class numbering : public smart_pointer<numbering_rep<T,M> > {
public:

// typedefs:

    typedef numbering_rep<T,M> rep;
    typedef smart_pointer<rep> base;
    typedef size_t              size_type;

// allocators:

    numbering (std::string name = "");
    numbering (numbering_rep<T,M>* ptr);
    ~numbering() {}

// accessors & modifiers:

    bool is_initialized() const { return base::operator->() != 0; }
    std::string name() const;
    size_type degree () const;
    void set_degree (size_type degree) const;
    bool is_continuous() const;
    bool is_discontinuous() const { return !is_continuous(); }
    bool has_compact_support_inside_element() const;
    const basis_basic<T>& get_basis() const { return base::data().get_basis(); }

    size_type      ndof      (const geo_size& gs, size_type map_dim) const;
    size_type dis_ndof      (const geo_size& gs, size_type map_dim) const;
    void          dis_idof   (const geo_size& gs, const geo_element& K, std::vector<size_
    void set_ios_permutations (const class geo_basic<T,M>& omega,
                                disarray<size_type,M>&      idof2ios_dis_idof,
                                disarray<size_type,M>&      ios_idof2dis_idof) const;

// comparator:

    bool operator== (const numbering<T,M>& y) const {
        if (! is_initialized() && ! y.is_initialized()) return true;
        if (! is_initialized() || ! y.is_initialized()) return false;
        return name() == y.name();
    }
// i/o:

    void dump(std::ostream& out = std::cerr) const;
};

```

6.5 basis_on_pointset - pre-evaluated polynomial basis

(Source file: 'nfem/gbasis/basis_on_pointset.h')

Synopsis

The `basis_on_pointset` class is able to evaluates a polynomial basis and its derivatives on a set of point of the reference element (see [reference_element iclass](#), page [undefined](#)). The basis is described by the `basis` class (see [basis class](#), page [undefined](#)). The set of points could be either quadrature nodes on the reference element (see [quadrature iclass](#), page [undefined](#)) or Lagrange nodes associated to another basis. For application of an integration of jump and average values of a discontinuous approximation on a side, the set of nodes could be defined on a specific side only. In all these cases, the evaluation of polynomials could be performed one time for all on the reference element and its result stored and reused for all elements of the mesh: the speedup is important, especially for high order polynomials.

Implementation

```
template<class T>
class basis_on_pointset: public smart_pointer<basis_on_pointset_rep<T> > {
public:

    typedef basis_on_pointset_rep<T>          rep;
    typedef smart_pointer<rep>                 base;
    typedef typename rep::size_type           size_type;
    typedef typename rep::const_iterator      const_iterator;
    typedef typename rep::const_iterator_grad const_iterator_grad;

    // allocators:

    basis_on_pointset ();
    basis_on_pointset (const quadrature<T>& quad, const basis_basic<T>& b);
    basis_on_pointset (const basis_basic<T>& nb,    const basis_basic<T>& b);

    // modifiers:

    void set (const quadrature<T>& quad, const basis_basic<T>& b);
    void set (const basis_basic<T>& nb,    const basis_basic<T>& b);

    // accessors:

    const basis_basic<T>& get_basis() const;
    size_type pointset_size (reference_element hat_K) const;

    // reference element and pointset:

    void evaluate      (reference_element hat_K, size_type q) const;
    void evaluate_grad (reference_element hat_K, size_type q) const;
```

```

// for a specific node instead of a pointset:
void evaluate      (reference_element hat_K, const point_basic<T>& hat_xq) const;
void evaluate_grad (reference_element hat_K, const point_basic<T>& hat_xq) const;

// side trace, for DG:
void restrict_on_side (reference_element tilde_L, const side_information_type& si

// accessors to values:

const T& value (size_type loc_idof) const;
const point_basic<T>& grad_value (size_type loc_idof) const;

const_iterator      begin() const;
const_iterator      end() const;
const_iterator_grad begin_grad() const;
const_iterator_grad end_grad() const;
};

```

6.6 Bk - Bernstein polynomial basis

(Source file: 'nfem/gbasis/basis_fem_Pk_bernstein.h')

Synopsis

```
space Vh(omega,"B5");
```

Description

This **basis** was initially introduced by Bernstein (Comm. Soc. Math. Kharkov, 2th series, 1912) and more recently used in the context of finite elements. It is indicated in the **space** (see [\[space class\]](#), page [\[undefined\]](#)) by a string starting with the letter "B", followed by digits indicating the polynomial order.

Options

This basis do not recognizes any option. See [\[basis_option class\]](#), page [\[undefined\]](#).

6.7 Pk - Lagrange polynomial basis

(Source file: 'nfem/gbasis/basis_fem_Pk_lagrange.h')

Synopsis

```
space Vh(omega,"P1");
```

Description

This is the most popular polynomial **basis** for the finite element method. It is indicated in the **space** (see [\[space class\]](#), page [\[undefined\]](#)) by a string starting with the letter "P", followed by digits indicating the polynomial order.

Options

This basis recognizes the `equispaced/warburton` node option and the raw polynomial option. See [\[basis_option class\]](#), page [\[undefined\]](#).

6.8 RTk - The Raviart-Thomas vector-valued polynomial basis

(Source file: `'nfem/gbasis/basis_fem_RTk.h'`)

Synopsis

```
space Vh(omega,"RT0");
```

Description

This **basis** is described in Raviart and Thomas (Mathematical aspects of finite element methods, Springer, 1977). It is indicated in the **space** (see [\[space class\]](#), page [\[undefined\]](#)) by a string starting with the two letters "RT", followed by digits indicating the polynomial order.

Options

This basis recognizes the `equispaced/warburton` node option for degrees of freedom located on sides. See [\[basis_option class\]](#), page [\[undefined\]](#).

6.9 Sk - Dubiner-Sherwin-Karniadakis polynomial basis

(Source file: `'nfem/gbasis/basis_fem_Pk_sherwin.h'`)

Synopsis

```
space Vh(omega,"S5");
```

Description

This **basis** is described for the triangle by Dubiner (J. Sci. Comput., 1991) and extended by Sherwin and Karniadakis (2005, Cambridge Univ. Press) to others reference elements. It is indicated in the **space** (see [\[space class\]](#), page [\[undefined\]](#)) by a string starting with the letter "S", followed by digits indicating the polynomial order.

Options

This basis recognizes the `equispaced/warburton` node option for degrees of freedom located on sides. See [\[basis_option class\]](#), page [\[undefined\]](#).

6.10 edge - Edge reference element

(Source file: 'nfem/geo_element/edge.icc')

Description

The edge reference element is $K = [0,1]$.

0-----1 x

Curved high order Pk edges ($k \geq 1$), in 2d or 3d geometries, are supported. These edges have internal nodes, numbered as:

0----2----1 0--2--3--1 0-2-3-4-1
 P2 P3 P4

Implementation

```
const size_t dimension = 1;
const Float   measure = 1;
const size_t n_vertex = 2;
const point vertex [n_vertex] = {
    point(0),
    point(1) };
```

6.11 geo_element - element of a mesh

(Source file: 'nfem/geo_element/geo_element_v4.h')

Description

Defines geometrical elements and sides as a set of vertice and edge indexes. This element is obtained after a Piola transformation from a reference element (see [reference_element](#) iclass, page [reference_element](#)). Indexes are related to arrays of edges and vertices. These arrays are included in the description of the mesh. Thus, this class is related of a given mesh instance (see [geo](#) class, page [geo](#)).

Example

This is the test of geo_element:

```
geo_element_auto<> K;
K.set_name('t') ;
cout << "n_vertices: " << K.size() << endl
      << "n_edges   : " << K.n_edges() << endl
      << "dimension : " << K.dimension() << endl << endl;
for(geo_element::size_type i = 0; i < K.size(); i++)
    K[i] = i*10 ;
for(geo_element::size_type i = 0; i < K.n_edges(); i++)
    K.set_edge(i, i*10+5) ;
cout << "vertices: local -> global" << endl;
for (geo_element::size_type vloc = 0; vloc < K.size(); vloc++)
```

```

        cout << vloc << "-> " << K[vloc] << endl;
    cout << endl
        << "edges: local -> global" << endl;
    for (geo_element::size_type eloc = 0; eloc < K.n_edges(); eloc++) {
        geo_element::size_type vloc1 = subgeo_local_vertex(1, eloc, 0);
        geo_element::size_type vloc2 = subgeo_local_vertex(1, eloc, 1);
        cout << eloc << "-> " << K.edge(eloc) << endl
            << "local_vertex_from_edge(" << eloc
            << ") -> (" << vloc1 << ", " << vloc2 << ")" << endl;
    }

```

6.12 hack_array - container in distributed environment

(Source file: 'nfem/geo_element/hack_array.h')

Synopsis

STL-like vector container for a distributed memory machine model. Contrarily to `disarray<T>`, here `T` can have a size only known at compile time. This class is used when `T` is a `geo_element` raw class, i.e. `T=geo_element_e.raw`. The size of the `geo_element` depends upon the order and is known only at run-time. For efficiency purpose, the `hack_array` allocate all `geo_elements` of the same variant (e.g. `edge`) and order in a contiguous area, since the corresponding element size is constant.

Example

A sample usage of the class is:

```

std::pair<size_t,size_t> param (reference_element::t, 3); // triangle, order=3
hack_array<geo_element_raw> x (distributor(100), param);

```

The `hack_array<T>` interface is similar to those of the `disarray<T>` one.

Object requirement

There are many pre-requisites for the template object type `T`:

```

class T : public T::generic_type {
    typedef variant_type;
    typedef raw_type;
    typedef genetic_type;
    typedef automatic_type;
    static const variant_type _variant;
    static size_t _data_size(const parameter_type& param);
    static size_t _value_size(const parameter_type& param);
};
class T::automatic_type : public T::generic_type {
    automatic_type (const parameter_type& param);
};
class T::generic_type {

```

```

typedef raw_type;
typedef iterator;
typedef const_iterator;
iterator _data_begin();
const_iterator _data_begin() const;
};
ostream& operator<< (ostream&, const T::generic_type&);

```

Implementation

```

template <class T, class A>
class hack_array<T,sequential,A> : public smart_pointer<hack_array_seq_rep<T,A>> {
public:

// typedefs:

    typedef hack_array_seq_rep<T,A>      rep;
    typedef smart_pointer<rep>           base;

    typedef sequential                   memory_type;
    typedef typename rep::size_type      size_type;
    typedef typename rep::value_type     value_type;
    typedef typename rep::reference      reference;
    typedef typename rep::dis_reference  dis_reference;
    typedef typename rep::iterator       iterator;
    typedef typename rep::const_reference const_reference;
    typedef typename rep::const_iterator const_iterator;
    typedef typename rep::parameter_type parameter_type;

// allocators:

    hack_array (const A& alloc = A());
    hack_array (size_type loc_size,          const parameter_type& param, const A&
    void resize (const distributor& ownership, const parameter_type& param);
    hack_array (const distributor& ownership, const parameter_type& param, const A&
    void resize (size_type loc_size,          const parameter_type& param);

// local accessors & modifiers:

    A get_allocator() const { return base::data().get_allocator(); }
    size_type size () const { return base::data().size(); }
    size_type dis_size () const { return base::data().dis_size(); }
    const distributor& ownership() const { return base::data().ownership(); }
    const communicator& comm() const { return ownership().comm(); }

    reference operator[] (size_type i) { return base::data().operator[]
    const_reference operator[] (size_type i) const { return base::data().operator[]

    const_reference dis_at (size_type dis_i) const { return base::data().operator[]

```



```

        iterator begin()          { return base::data().begin(); }
const_iterator begin() const { return base::data().begin(); }
        iterator end()            { return base::data().end(); }
const_iterator end() const      { return base::data().end(); }

// global accessors (for compatibility with distributed interface):

template<class Set> void append_dis_indexes (const Set& ext_idx_set) const {}
void update_dis_entries() const {}

// global modifiers (for compatibility with distributed interface):

dis_reference dis_entry (size_type dis_i) { return operator[] (dis_i); }
void dis_entry_assembly() {}
template<class SetOp>
void dis_entry_assembly(SetOp my_set_op) {}
template<class SetOp>
void dis_entry_assembly_begin (SetOp my_set_op) {}
template<class SetOp>
void dis_entry_assembly_end (SetOp my_set_op) {}

// apply a partition:

#ifdef TODO
template<class RepSize>
void repartition (
    const RepSize&          partition,          // old_numbering for *this
    hack_array<T,sequential,A>& new_array,      // old_ownership
    RepSize&                old_numbering,      // new_ownership (created)
    RepSize&                new_numbering,      // new_ownership
    const RepSize&          new_array,          // old_ownership
    { return base::data().repartition (partition, new_array, old_numbering, new

template<class RepSize>
void permutation_apply (
    const RepSize&          new_numbering,      // old_numbering for *this
    hack_array<T,sequential,A>& new_array) const // old_ownership
    { return base::data().permutation_apply (new_numbering, new_array); }
#endif // TODO

// i/o:

odiststream& put_values (odiststream& ops) const { return base::data().put_valu
idiststream& get_values (idiststream& ips)      { return base::data().get_valu
template <class GetFunction>
idiststream& get_values (idiststream& ips, GetFunction get_element) { ret
template <class PutFunction>
odiststream& put_values (odiststream& ops, PutFunction put_element) const { ret
#endif TODO

```

```

        void dump (std::string name) const { return base::data().dump(name); }
    #endif // TODO
};

```

Implementation

```

template <class T, class A>
class hack_array<T,distributed,A> : public smart_pointer<hack_array_mpi_rep<T,A> >
public:

    // typedefs:

    typedef hack_array_mpi_rep<T,A>      rep;
    typedef smart_pointer<rep>           base;

    typedef distributed                  memory_type;
    typedef typename rep::size_type     size_type;
    typedef typename rep::value_type    value_type;
    typedef typename rep::reference     reference;
    typedef typename rep::dis_reference dis_reference;
    typedef typename rep::iterator      iterator;
    typedef typename rep::parameter_type parameter_type;
    typedef typename rep::const_reference const_reference;
    typedef typename rep::const_iterator const_iterator;
    typedef typename rep::scatter_map_type scatter_map_type;

    // allocators:

    hack_array (const A& alloc = A());
    hack_array (const distributor& ownership, const parameter_type& param, const A&
    void resize (const distributor& ownership, const parameter_type& param);

    // local accessors & modifiers:

    A get_allocator() const { return base::data().get_allocator(); }
    size_type size () const { return base::data().size(); }
    size_type dis_size () const { return base::data().dis_size(); }
    const distributor& ownership() const { return base::data().ownership(); }
    const communicator& comm() const { return base::data().comm(); }

    reference operator[] (size_type i) { return base::data().operator[]
    const_reference operator[] (size_type i) const { return base::data().operator[]

        iterator begin() { return base::data().begin(); }
    const_iterator begin() const { return base::data().begin(); }
        iterator end() { return base::data().end(); }
    const_iterator end() const { return base::data().end(); }

    // global accessor:

```

```

template<class Set, class Map>
void append_dis_entry (const Set& ext_idx_set, Map& ext_idx_map) const { base::

template<class Set, class Map>
void get_dis_entry    (const Set& ext_idx_set, Map& ext_idx_map) const { base::

template<class Set>
void append_dis_indexes (const Set& ext_idx_set) const { base::data().append_di

template<class Set>
void set_dis_indexes   (const Set& ext_idx_set) { base::data().set_dis_indexe

const_reference dis_at (size_type dis_i) const { return base::data().dis_at (di

// get all external pairs (dis_i, values):
const scatter_map_type& get_dis_map_entries() const { return base::data().get_d

void update_dis_entries() const { base::data().update_dis_entries(); }

// global modifiers (for compatibility with distributed interface):

dis_reference dis_entry (size_type dis_i)          { return base::data().dis_entry

void dis_entry_assembly()                          { return base::data().dis_entry

template<class SetOp>
void dis_entry_assembly      (SetOp my_set_op) { return base::data().dis_entry
template<class SetOp>
void dis_entry_assembly_begin (SetOp my_set_op) { return base::data().dis_entry
template<class SetOp>
void dis_entry_assembly_end   (SetOp my_set_op) { return base::data().dis_entry

// apply a partition:

template<class RepSize>
void repartition (
    const RepSize&          partition,          // old_numbering for *this
    const RepSize&          new_numbering,       // old_ownership
    hack_array<T,distributed>& new_array,        // new_ownership (created)
    const RepSize&          old_numbering,       // new_ownership
    const RepSize&          new_numbering) const // old_ownership
{ return base::data().repartition (partition.data(), new_array.data(), old_

#ifdef TODO
template<class RepSize>
void permutation_apply (
    const RepSize&          new_numbering,       // old_numbering for *this
    const RepSize&          old_numbering,       // old_ownership
    hack_array<T,distributed,A>& new_array) const // new_ownership (already a
{ base::data().permutation_apply (new_numbering.data(), new_array.data()); }

```

```

        void reverse_permutation (                                // old_ownership for
            hack_array<size_type,distributed,A>& inew2dis_iold) const // new_ownership
        { base::data().reverse_permutation (inew2dis_iold.data()); }
#endif // TODO

// i/o:

    odistream& put_values (odistream& ops) const { return base::data().put_values(ops); }
    idistream& get_values (idistream& ips)       { return base::data().get_values(ips); }
#ifdef TODO
    void dump (std::string name) const           { return base::data().dump(name); }
#endif // TODO

template <class GetFunction>
idistream& get_values (idistream& ips, GetFunction get_element)
    { return base::data().get_values(ips, get_element); }
template <class PutFunction>
odistream& put_values (odistream& ops, PutFunction put_element) const
    { return base::data().put_values(ops, put_element); }

template <class PutFunction, class Permutation>
odistream& permuted_put_values (
    odistream& ops,
    const Permutation& perm,
    PutFunction put_element) const
    { return base::data().permuted_put_values (ops, perm.data(), put_element); }
};

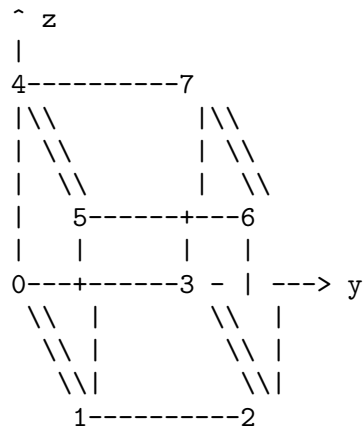
```

6.13 hexahedron - Hexaedron reference element

(Source file: 'nfem/geo_element/hexahedron.icc')

Description

The hexahedron reference element is $[-1,1]^3$.



\\
x

Curved high order Pk hexaedra ($k \geq 1$) in 3d geometries are supported. These hexaedra have additional edge-nodes, face-nodes and internal volume-nodes.

These nodes are numbered as

first vertex, then edge-node, following the edge numbering order and orientation, then face-nodes following the face numbering order and orientation, and finally the face internal nodes, following the hexaedron lattice. See below for edges and faces numbering and orientation.

```

4----19----7
|\\      |\\
|16      23 | 18
12  \\ 21    15  \\
|    5----17+---6
|22 | 26 | 25|
0---+-11---3 |
\\ 13      24  \\ 14
8 | 20      10|
\\|      \\|
1-----9----2
P2

```

Numbering

The orientation is such that trihedra (01, 03, 04) is direct and all faces, seen from exterior, are in the direct sense. References: P. L. Georges, "Generation automatique de maillages", page 24-, coll RMA, 16, Masson, 1994. Notice that the edge-nodes and face-nodes numbering slightly differ from those used in the `gmsh` mesh generator when using high-order elements. This difference is handled by the `msh2geo` mesh file converter (see `<undefined>` [msh2geo command], page `<undefined>`).

Implementation

```

const size_t dimension = 3;
const Float  measure = 8;
const size_t n_vertex = 8;
const point vertex [n_vertex] = {
    point(-1,-1,-1 ),
    point( 1,-1,-1 ),
    point( 1, 1,-1 ),
    point(-1, 1,-1 ),
    point(-1,-1, 1 ),
    point( 1,-1, 1 ),
    point( 1, 1, 1 ),
    point(-1, 1, 1 ) };
const size_t n_face = 6;
const size_t face [n_face][4] = {

```

```

        {0, 3, 2, 1 },
        {0, 4, 7, 3 },
        {0, 1, 5, 4 },
        {4, 5, 6, 7 },
        {1, 2, 6, 5 },
        {2, 3, 7, 6 } };
const size_t n_edge = 12;
const size_t edge [n_edge][2] = {
    {0, 1 },
    {1, 2 },
    {2, 3 },
    {3, 0 },
    {0, 4 },
    {1, 5 },
    {2, 6 },
    {3, 7 },
    {4, 5 },
    {5, 6 },
    {6, 7 },
    {7, 4 } };

```

6.14 point - Point reference element

(Source file: 'nfem/geo_element/point.icc')

Description

The point reference element is defined for convenience. It is a 0-dimensional element with measure equal to 1.

Implementation

```

const size_t dimension = 0;
const Float measure = 1;

```

6.15 prism - Prism reference element

(Source file: 'nfem/geo_element/prism.icc')

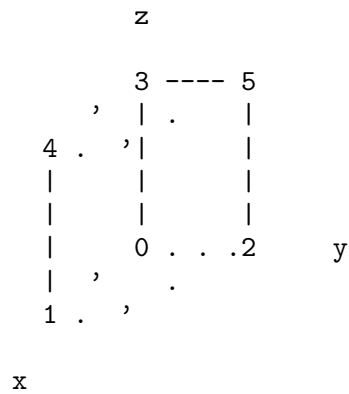
Description

The prism reference element is

$$K = \{ 0 < x < 1 \text{ and } 0 < y < 1-x \text{ and } -1 < z < 1 \}$$

Numbering

The orientation is such that trihedra (01, 02, 03) is direct and all faces, see from exterior, are in the direct sens. References: P. L. Georges, "Generation automatique de maillages", page 24-, coll RMA, 16, Masson, 1994.



Implementation

```

const size_t dimension = 3;
const Float  measure = 1;
const size_t n_vertex = 6;
const point vertex [n_vertex] = {
    point( 0, 0,-1 ),
    point( 1, 0,-1 ),
    point( 0, 1,-1 ),
    point( 0, 0, 1 ),
    point( 1, 0, 1 ),
    point( 0, 1, 1 ) };
const size_t  n_face = 5;
const size_t face [n_face][4] = {
    { 0, 2, 1, size_t(-1) },
    { 3, 4, 5, size_t(-1) },
    { 0, 1, 4, 3 },
    { 1, 2, 5, 4 },
    { 0, 3, 5, 2 } };
const size_t  n_edge = 9;
const size_t edge [n_edge][2] = {
    { 0, 1 },
    { 1, 2 },
    { 2, 0 },
    { 0, 3 },
    { 1, 4 },
    { 2, 5 },
    { 3, 4 },
    { 4, 5 },
    { 5, 3 } };

```

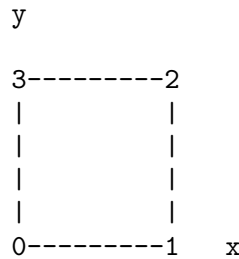
6.16 quadrangle - Quadrangular reference element

(Source file: 'nfem/geo_element/quadrangle.icc')

Description

The quadrangular reference element is $[-1,1]^2$.

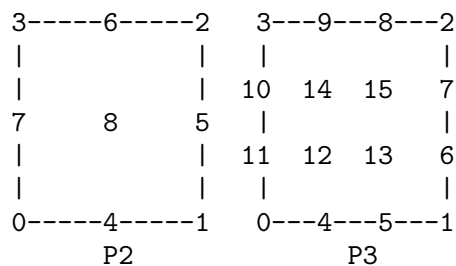
Numbering



Curved high order P_k quadrangles ($k \geq 1$), in 2d or 3d geometries, are supported. These quadrangles have additional edge-nodes and face-nodes.

These nodes are numbered as

first vertex, then edge-node, following the edge numbering order and orientation, and finally the face internal nodes, following the quadrangle lattice. See below for edge numbering and orientation.



Implementation

```

const size_t dimension = 2;
const Float  measure = 4;
const size_t n_vertex = 4;
const point vertex [n_vertex] = {
    point(-1,-1),
    point( 1,-1),
    point( 1, 1),
    point(-1, 1) };
const size_t  n_edge = 4;
const size_t edge [n_edge][2] = {
    { 0, 1 },
    { 1, 2 },
    { 2, 3 },
    { 3, 0 } };
  
```


6.17 quadrature - quadrature formulae on the reference element

(Source file: 'nfem/geo_element/quadrature.h')

Synopsis

The `quadrature` class defines a container for a quadrature formulae on the reference element (see `<reference_element_iclass>`, page `<undefined>`). This container stores the nodes coordinates and the weights.

The constructor takes two arguments

the reference element K and the order r of the quadrature formulae. The formulae is exact when computing the integral of a polynom p that degree is less or equal to order r .

$$\int_K p(x) \, dx = \sum_{q=1}^n p(x_q) w_q$$

Limitations

The formulae is optimal when it uses a minimal number of nodes n . Optimal quadrature formula are hard-coded in this class. Not all reference elements and orders are yet implemented. This class will be completed in the future.

Implementation

```
template<class T>
class quadrature : public smart_pointer<quadrature_rep<T> > {
public:

// typedefs:

    typedef quadrature_rep<T>          rep;
    typedef smart_pointer<rep>         base;
    typedef typename rep::size_type    size_type;
    typedef typename rep::family_type  family_type;
    typedef typename rep::const_iterator const_iterator;
    typedef typename rep::orientation_type orientation_type;

// allocators:

    quadrature (quadrature_option opt = quadrature_option())
        : base(new_macro(rep(opt))) {}

// modifiers:

    void set_order (size_type order) { base::data().set_order(order); }
```

```

        void set_family (family_type ft)  { base::data().set_family(ft); }

// accessors:

    size_type      get_order() const { return base::data().get_order();}
    family_type    get_family() const { return base::data().get_family();}
    std::string    get_family_name() const { return base::data().get_family_name();}
    const quadrature_option& get_options() const { return base::data().get_options();}
    size_type      size (reference_element hat_K) const { return base::data().size(hat_K);}
    const_iterator begin (reference_element hat_K) const { return base::data().begin(hat_K);}
    const_iterator end   (reference_element hat_K) const { return base::data().end(hat_K);}
    const weighted_point<T>& operator() (reference_element hat_K, size_type q) const { return base::data().operator()(hat_K, q);}

// side accessor:

    void side_initialize (
        reference_element      tilde_K,
        size_type              loc_isid,
        reference_element      hat_S,
        size_type              shift,
        orientation_type        orient) const {
        base::data().side_initialize (tilde_K, loc_isid, hat_S, shift, orient);
    }
};

```

6.18 reference_element - reference element

(Source file: 'nfem/geo_element/reference_element.h')

Synopsis

The `reference_element` class defines all supported types of geometrical elements in one, two and three dimensions. The set of supported elements are designate by a letter

'p'	point (dimension 0)
'e'	edge (dimension 1)
't'	triangle(dimension 2)
'q'	quadrangle(dimension 2)
'T'	tetrahedron(dimension 3)
'P'	prism(dimension 3)
'H'	hexaedron(dimension 3)

Implementation

```

class reference_element {

```

```

public:

// typedefs:

    typedef std::vector<int>::size_type size_type;

    // defines variant_type { p, t, q ..., H, ...};
    // in an automatically generated file :

    typedef size_type variant_type;
    static const variant_type
        p = 0,
        e = 1,
        t = 2,
        q = 3,
        T = 4,
        P = 5,
        H = 6,
        max_variant = 7;

// allocators/deallocators:

    reference_element (variant_type x = max_variant)
        : _x(x) { assert_macro (x >= 0 && x <= max_variant, "invalid type " << x); }

// initializers:

    void set_variant (variant_type x) { _x = x; }
    void set_variant (size_type n_vertex, size_type dim) { _x = variant (n_vertex, dim); }
    void set_name (char name);

// accessors:

    variant_type variant() const { return _x; }
    char name() const { return _name[_x % max_variant]; }
    size_type dimension() const { return _dimension[_x]; }
    size_type size() const { return _n_vertex[_x]; }
    size_type n_subgeo(size_type subgeo_dim) const { return n_subgeo (variant(), subgeo_dim); }
    size_type n_edge() const { return n_subgeo(1); }
    size_type n_face() const { return n_subgeo(2); }
    size_type subgeo_size (size_type subgeo_dim, size_type loc_isid) const {
        return subgeo_n_node (_x, 1, subgeo_dim, loc_isid); }
    size_type subgeo_local_vertex(size_type subgeo_dim, size_type loc_isid, size_type loc_jsid) const {
        return subgeo_local_node (_x, 1, subgeo_dim, loc_isid, loc_jsid); }

    // TODO: use template<class T> instead of Float
    const point_basic<Float>& vertex (size_type iloc) const;
    friend Float measure (reference_element hat_K);
    Float side_measure (size_type loc_isid) const;

```

```

    void side_normal (size_type loc_isid, point_basic<Float>& hat_n) const;

// helpers:

    static variant_type variant    (char name);
    static variant_type variant    (size_type n_vertex, size_type dim);
    static char          name      (variant_type variant) { return _name      [variant]; }
    static size_type dimension    (variant_type variant) { return _dimension[variant]; }
    static size_type n_vertex     (variant_type variant) { return _n_vertex  [variant]; }
    static size_type n_node       (variant_type variant, size_type order);

    static size_type n_sub_edge    (variant_type variant);
    static size_type n_sub_face    (variant_type variant);
    static size_type n_subgeo      (variant_type variant, size_type subgeo_dim);
    static size_type subgeo_n_node (variant_type variant, size_type order, size_type subgeo_dim);
    static size_type subgeo_local_node (variant_type variant, size_type order, size_type subgeo_dim);

    static variant_type first_variant_by_dimension (size_type dim) {
        return _first_variant_by_dimension[dim]; }
    static variant_type last_variant_by_dimension (size_type dim) {
        return _first_variant_by_dimension[dim+1]; }

    static size_type first_inod_by_variant (variant_type variant, size_type order, size_type subgeo_dim);
    static size_type last_inod_by_variant (variant_type variant, size_type order, size_type subgeo_dim);
    static size_type first_inod_by_variant (variant_type variant, size_type order, size_type subgeo_dim, size_type subgeo_dim+1); }
    static size_type first_inod (variant_type variant, size_type order, size_type subgeo_dim);
    static size_type last_inod (variant_type variant, size_type order, size_type subgeo_dim);
    static size_type first_inod (variant_type variant, size_type order, size_type subgeo_dim, size_type subgeo_dim+1); }
    static size_type last_inod (variant_type variant, size_type order, size_type subgeo_dim, size_type subgeo_dim+1); }
    static void init_local_nnode_by_variant (size_type order, std::array<size_type, 5> & nnode);

protected:
// constants:

    static const char      _name [max_variant];
    static const size_type _dimension [max_variant];
    static const size_type _n_vertex [max_variant];
    static const variant_type _first_variant_by_dimension[5];

// data:

    variant_type _x;
};

```

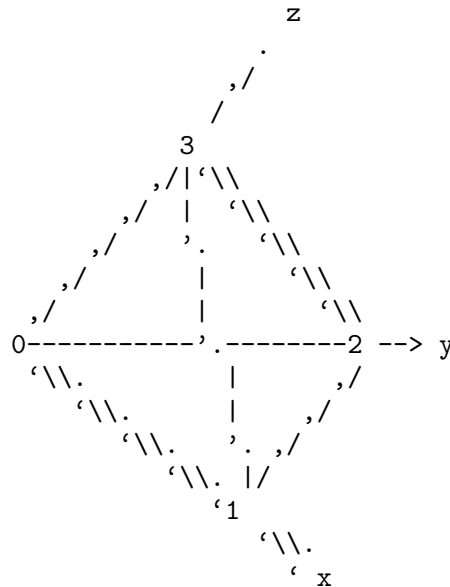
6.19 tetrahedron - Tetraedron reference element

(Source file: 'nfem/geo_element/tetrahedron.icc')

Description

The tetrahedron reference element is

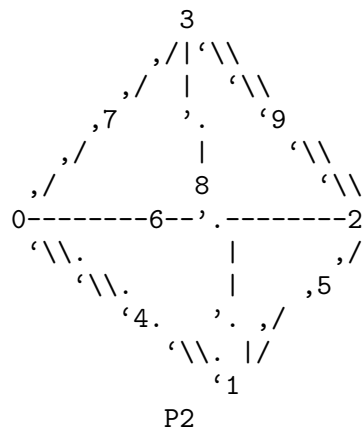
$$K = \{ 0 < x < 1 \text{ and } 0 < y < 1-x \text{ and } 0 < z < 1-x-y \}$$



Curved high order Pk tetrahedra ($k \geq 1$) in 3d geometries are supported. These tetrahedra have additional edge-nodes, face-nodes and internal volume-nodes.

These nodes are numbered as

first vertex, then edge-node, following the edge numbering order and orientation, then face-nodes following the face numbering order and orientation, and finally the face internal nodes, following the tetrahedron lattice. See below for edges and faces numbering and orientation.



Numbering

The orientation is such that triad (01, 02, 03) is direct, and all faces, seen from exterior, are in the direct sense. References: P. L. Georges, "Generation automatique de maillages", page

24-, coll RMA, 16, Masson, 1994. Notice that the edge-nodes and face-nodes numbering slightly differ from those used in the `gms` mesh generator when using high-order elements. This difference is handled by the `msh2geo` mesh file converter (see `[msh2geo command]`, page `<undefined>`).

Implementation

```
const size_t dimension = 3;
const Float  measure = Float(1.)/Float(6.);
const size_t n_vertex = 4;
const point vertex [n_vertex] = {
    point( 0, 0, 0 ),
    point( 1, 0, 0 ),
    point( 0, 1, 0 ),
    point( 0, 0, 1 ) };
const size_t  n_face = 4;
const size_t face [n_face][3] = {
    { 0, 2, 1 },
    { 0, 3, 2 },
    { 0, 1, 3 },
    { 1, 2, 3 } };
const size_t  n_edge = 6;
const size_t edge [n_edge][2] = {
    { 0, 1 },
    { 1, 2 },
    { 2, 0 },
    { 0, 3 },
    { 1, 3 },
    { 2, 3 } };
```

6.20 triangle - Triangle reference element

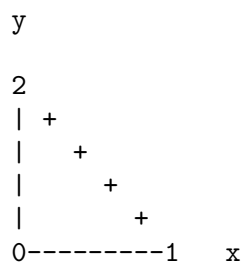
(Source file: `'nfem/geo_element/triangle.icc'`)

Description

The triangle reference element is

$$K = \{ 0 < x < 1 \text{ and } 0 < y < 1-x \}$$

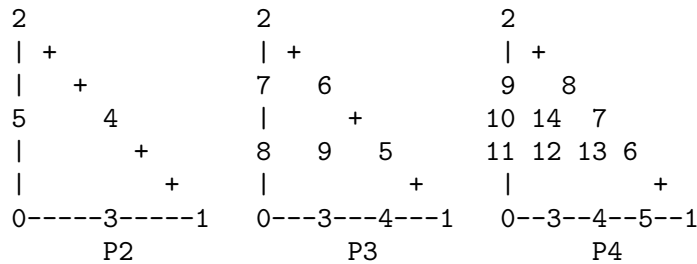
Numbering



Curved high order Pk triangles ($k \geq 1$), in 2d or 3d geometries, are supported. These triangles have additional edge-nodes and face-nodes.

These nodes are numbered as

first vertex, then edge-node, following the edge numbering order and orientation, and finally the face internal nodes, following the triangle lattice. See below for edge numbering and orientation.



Implementation

```
const size_t dimension = 2;
const Float  measure = 0.5;
const size_t n_vertex = 3;
const point vertex [n_vertex] = {
    point(0, 0),
    point(1, 0),
    point(0, 1) };
const size_t n_edge = 3;
const size_t edge [n_edge][2] = {
    { 0, 1 },
    { 1, 2 },
    { 2, 0 } };
```

6.21 dis_cpu_time, dis_wall_time, seq_cpu_time, seq_walltime – Time in seconds since an arbitrary time in the past.

(Source file: 'skit/plib2/dis_cpu_time.h')

Synopsis

```
double dis_cpu_time();
double dis_wall_time();
double seq_cpu_time();
double seq_wall_time();
```

Example

```
double t_start = dis_wall_time();
//.... stuff to be timed ...
```

```
double t_end    = dis_wall_time();
derr << "That took " << t_end - t_start << " seconds" << endl

double cpu_start = dis_cpu_time();
//.... stuff to be timed ...
double cpu_end   = dis_cpu_time();
derr << "That needs " << cpu_end - cpu_start << " CPU seconds" << endl
```

Description

`dis_wall_time` returns a floating-point number of seconds, representing elapsed wall-clock time since some time in the past. The *time in the past* is guaranteed not to change during the life of the process. The user is responsible for converting large numbers of seconds to other units if they are preferred. This function is portable (it returns seconds, not *ticks*), it allows high resolution, and carries no unnecessary baggage. In a distributed environment, `dis_wall_time` clocks are synchronized: different nodes return the same time value at the same instant. `seq_wall_time` is similar but the time returned is local to the node that called them and clocks are not synchronized: in a distributed environment, different nodes can return different local times, at different instant when the call to `seq_wall_time` is reached.

`seq_cpu_time` is similar but returns the computing time per user for the current process. `dis_cpu_time` returns the accumulated CPU for all processed linked together via the default communicator.

6.22 `dis_memory_usage`, `seq_memory_usage` – physical memory in use

(Source file: ‘skit/plib2/dis_memory_usage.h’)

Synopsis

```
size_t dis_memory_usage();
size_t seq_memory_usage();
size_t dis_peak_memory_usage();
size_t seq_peak_memory_usage();
size_t memory_size();
```

Description

`seq_memory_usage` returns the current resident set size (physical memory use) measured in bytes, or zero if the value cannot be determined on this OS.

`dis_memory_usage` returns the accumulated memory usage for all processed linked together via the default communicator.

`seq_peak_memory_usage` returns the peak (maximum so far) resident set size (physical memory use) measured in bytes, or zero if the value cannot be determined on this OS.

`dis_peak_memory_usage` returns the accumulated peak memory usage for all processed linked together via the default communicator.

`memory_size` returns the size of physical memory (RAM) in bytes associated to the given process.

6.23 `index_set` - a set of indexes

(Source file: `'index_set_header.icc'`)

Synopsis

A class for: `l = {1,3,...9}` i.e. a wrapper for STL `set<size_t>` with some assignment operators, such as `l1 += l2`. This class is suitable for use with the `disarray<T>` class, as `disarray<index_set>` (see [\[disarray class\]](#), page [\[undefined\]](#)).

Implementation

```
class index_set : public std::set<std::size_t> {
public:

// typedefs:

    typedef std::set<std::size_t>    base;
    typedef std::size_t              value_type;
    typedef std::size_t              size_type;

// allocators:

    index_set ();
    index_set (const index_set& x);
    index_set& operator= (const index_set& x);
    template <int N>
    index_set& operator= (size_type x[N]);
    void clear ();

// basic algebra:

    void insert (size_type dis_i);    // a := a union {dis_i}
    index_set& operator+= (size_type dis_i);    // idem
    index_set& operator+= (const index_set& b); // a := a union b

// a := a union b
    void inplace_union (const index_set& b);
    void inplace_intersection (const index_set& b);

// c := a union b
    friend void set_union (const index_set& a, const index_set& b, index_set& c);
    friend void set_intersection (const index_set& a, const index_set& b, index_set& c);
```

```

// io:

    friend std::istream& operator>> (std::istream& is, index_set& x);
    friend std::ostream& operator<< (std::ostream& os, const index_set& x);

// boost mpi:

#ifdef _RHEOLEF_INDEX_SET_H
#ifdef _RHEOLEF_HAVE_MPI
    template <class Archive>
        void serialize (Archive& ar, const unsigned int version);
#endif // _RHEOLEF_HAVE_MPI
#endif // _RHEOLEF_INDEX_SET_H
};

```

6.24 pair_set - a set of (index,value) pair

(Source file: 'skit/plib2/pair_set.h')

Synopsis

A class for: $l = \{(0,3.3), \dots (7,8.2)\}$ i.e. a wrapper for `STL map<size_t,T>` with some assignment operators, such as `l1 += l2`. This class is suitable for use with the `disarray<T>` class, as `disarray<pair_set>` (see [\(undefined\)](#) [disarray class], page [\(undefined\)](#)).

Todo

template <T,A> with A=std::allocator or heap_allocator

Implementation

```

template<class T, class A = std::allocator<std::pair<std::size_t,T> > >
class pair_set: public std::map<std::size_t, T, std::less<std::size_t>, A> {
public:

// typedefs:

    typedef std::size_t                size_type;
    typedef std::pair<size_type,T>     pair_type;
    typedef std::pair<const size_type,T> const_pair_type;
    typedef pair_type                  value_type;
    typedef A                          allocator_type;
    typedef std::map<size_type, T, std::less<size_type>, allocator_type>
                                      base;

    typedef typename base::iterator    iterator;
    typedef typename base::const_iterator const_iterator;

// allocators:

```

```

pair_set (const A& alloc = A());
pair_set (const pair_set<T,A>& x, const A& alloc = A());
pair_set<T,A>& operator= (const pair_set<T,A>& x);
void clear ();

// basic algebra: semantic of a sparse vector

pair_set<T,A>& operator+= (const pair_type&      x); // c := a union {x}
template<class B>
pair_set<T,A>& operator+= (const pair_set<T,B>& b); // c := a union b

// boost mpi:

template <class Archive>
void serialize (Archive& ar, const unsigned int version);
};
// io:
template <class T, class A>
std::istream& operator>> (std::istream& is,      pair_set<T,A>& a);
template <class T, class A>
std::ostream& operator<< (std::ostream& os, const pair_set<T,A>& b);

```

6.25 Vector - STL vector<T> with reference counting

(Source file: 'util/lib/Vector.h')

Description

The class implement a reference counting wrapper for the STL `vector<T>` container class, with shallow copies. See also: *The standard template library*, by Alexander Stephanov and Meng Lee.

This class provides the full `vector<T>` interface specification an could be used instead of `vector<T>`.

Note

The write accessors

```
T& operator[] (size_type)
```

as in `v[i]` may checks the reference count for each access. For a loop, a better usage is:

```

Vector<T>::iterator i = v.begin();
Vector<T>::iterator last = v.end();
while (i != last) { ...}

```

and the reference count check step occurs only two time, when accessing via `begin()` and `end()`.

Thus, in order to encourage users to do it, we declare private theses member functions. A synonym of `operator[]` is at.

Implementation

```

template<class T>
class Vector : private smart_pointer<vector_rep<T> > {

public:

// typedefs:

    typedef iterator;
    typedef const_iterator;
    typedef pointer;
    typedef reference;
    typedef const_reference;
    typedef size_type;
    typedef difference_type;
    typedef value_type;
    typedef reverse_iterator;
    typedef const_reverse_iterator;

// allocation/deallocation:

    explicit Vector (size_type n = 0, const T& value = T ());
    Vector (const_iterator first, const_iterator last);
    void reserve (size_type n);
    void swap (Vector<T>& x) ;

// accessors:

    iterator          begin ();
    const_iterator    begin () const;
    iterator          end ();
    const_iterator    end ()   const;
    reverse_iterator  rbegin();
    const_reverse_iterator  rbegin() const;
    reverse_iterator  rend();
    const_reverse_iterator  rend() const;
    size_type size ()const;
    size_type max_size () const;
    size_type capacity () const;
    bool empty () const;
    void resize (size_type sz, T v = T ()); // non-standard ?
private:
    const_reference operator[] (size_type n) const;
    reference operator[] (size_type n);
public:
    const_reference at (size_type n) const; // non-standard ?
    reference at (size_type n);
    reference          front ();

```

```

        const_reference    front () const;
        reference          back ();
        const_reference    back () const;

// insert/erase:

        void push_back (const T& x);
        iterator insert (iterator position, const T& x = T ());
        void insert (iterator position, size_type n, const T& x);
        void insert (iterator position, const_iterator first, const_iterator last);
        void pop_back ();
        void erase (iterator position);
        void erase (iterator first, iterator last);
};

```

6.26 heap_allocator - heap-based allocator

(Source file: 'util/lib/heap_allocator.h')

Description

Heap allocators are generally used when there is a lot of allocation and deallocation of small objects. For instance, this is often the case when dealing with `std::list` and `std::map`.

Heap-based allocator is conform to the STL specification of allocators. It does not "free" the memory until the heap is destroyed.

This allocator handles an a priori unlimited area of memory: a sequence of growing chunks are allocated. For a limited memory handler in the same spirit, see "stack_allocator"(9).

Example

```

typedef map <size_t, double, less<size_t>, heap_allocator<pair<size_t,double> >
map_type a;
a.insert (make_pair (0, 3.14));
a.insert (make_pair (1, 1.17));
for (map_type::iterator iter = a.begin(), last = a.end(); iter != last; iter++)
    cout << (*iter).first << " " << (*iter).second << endl;
}

```

Implementation

```

template <typename T>
class heap_allocator {
protected:
    struct handler_type; // forward declaration:
public:

// typedefs:

```

```

typedef size_t      size_type;
typedef std::ptrdiff_t difference_type;
typedef T*          pointer;
typedef const T*    const_pointer;
typedef T&          reference;
typedef const T&    const_reference;
typedef T           value_type;

// constructors:

heap_allocator() throw()
    : handler (new handler_type)
{
}
heap_allocator (const heap_allocator& ha) throw()
    : handler (ha.handler)
{
    ++handler->reference_count;
}
template <typename U>
heap_allocator (const heap_allocator<U>& ha) throw()
    : handler ((typename heap_allocator<T>::handler_type*)(ha.handler))
{
    ++handler->reference_count;
}
~heap_allocator() throw()
{
    check_macro (handler != NULL, "unexpected null mem_info");
    if (--handler->reference_count == 0) delete handler;
}
// Rebind to allocators of other types
template <typename U>
struct rebind {
    typedef heap_allocator<U> other;
};

// assignment:

heap_allocator& operator= (const heap_allocator& ha)
{
    handler = ha.handler;
    ++handler->reference_count;
    return *this;
}

// utility functions:

pointer      address (reference r)      const { return &r; }

```

```

const_pointer address (const_reference c) const { return &c; }
size_type          max_size() const { return std::numeric_limits<size_t>::max() / si

// in-place construction/destruction

void construct (pointer p, const_reference c)
{
    // placement new operator:
    new( reinterpret_cast<void*>(p) ) T(c);
}
// C++ 2011: default construct a value of type T at the location referenced by p
void construct (pointer p) { new ( reinterpret_cast<void*>(p) ) T(); }

void destroy (pointer p)
{
    // call destructor directly:
    (p)->~T();
}

// allocate raw memory

pointer allocate (size_type n, const void* = NULL)
{
    return pointer (handler->raw_allocate (n*sizeof(T)));
}
void deallocate (pointer p, size_type n)
{
    // No need to free heap memory
}
const handler_type* get_handler() const {
    return handler;
}

// data:

protected:
    handler_type* handler;
    template <typename U> friend class heap_allocator;
};

```

6.27 smart_pointer, smart_pointer_clone - reference counted safe pointer with true copy semantic

(Source file: 'util/lib/smart_pointer.h')

Description

Here is a convenient way to implement a true copy semantic, by using shallow copies and reference counting, in order to minimise memory copies. This concept is generally related to the *smart pointer* method for managing memory.

The true semantic copy is defined as follows: if an object *A* is assigned to *B*, such as *A = B*, every further modification on *A* or *B* does not modify the other.

Notice that this class differs from the `boost::shared_ptr` class that implements safe pointers without the true copy semantic.

Clone variant

The `smart_pointer_clone` variant uses a `T* T::clone() const` member function instead of the usual `T::T()` copy constructor for obtaining a true copy of the data. This variant is motivated as follows: when using hierarchies of derived classes (also known as polymorphic classes), the usual copy is not possible because `c++` copy constructors cannot be virtual, so you cannot make a copy this way. This is a well-known problem with `C++`'s implementation of polymorphism.

We use a solution to the non-virtual copy constructor problem which is suggested by Ellis and Stroustrup in "The Annotated LRM". The solution is to require the *T* class to provide a virtual clone method for every class which makes a copy using `new` and the correct copy constructor, returning the result as a pointer to the superclass *T*. Each subclass of *T* overloads this function with its own variant which copies its own type. Thus the copy operation is now virtual and furthermore is localised to the individual subclass.

Nocopy variant

This variant of the smart pointer is designed for use on objects that cannot (or must not) be copied. An example would be when managing an object that contains, say, a file handle. It is essential that this not be copied because then you get the problem of deciding which copy is responsible for closing the file. To avoid the problem, wrap the file handle in a class and then manage a unique instance of it using a `smart_pointer_nocopy`. This ensures that the file handle cannot be copied and is closed when the last alias is destroyed.

The interface to the nocopy variant is the same as `smart_pointer` but with all operations that perform copying forbidden. In fact, because all three variants are instances of a common superclass, the forbidden methods do exist but will cause an error and exit if they are called.

The following modifiers cannot be used because they use copying of the pointed-to object and will therefore cause an error:

```
T* operator-> ();
T& operator* ();
T* pointer ();
T& data ();
```

References

- [1] A. Geron and F. Tawbi,
Pour mieux developper avec C++ : design pattern, STL, RTTI et smart pointers,
InterEditions, 1999. Page 118.
- [2] STLplus, http://stlplus.sourceforge.net/stlplus3/docs/smart_ptr.html
for the clone and nocopy variants.

Implementation

```

template <class T, class C>
class smart_pointer_base {
public:

    struct internal {};

// allocators:

    smart_pointer_base (T* p = 0);
    smart_pointer_base (const smart_pointer_base<T,C>&);
    smart_pointer_base (void* count, internal);
    smart_pointer_base<T,C>& operator= (const smart_pointer_base<T,C>&);
    ~smart_pointer_base ();

// accessors:

    const T* pointer      () const;
    const T& data         () const;
    const T* operator->   () const;
    const T& operator*    () const;

// modifiers:

    T* pointer      ();
    T& data         ();
    T* operator->   ();
    T& operator*    ();

// implementation:
private:
    struct counter {
        T* _p;
        int _n;
        counter (T* p = 0);
        ~counter ();
        int operator++ ();
        int operator-- ();
    };
    counter *_count;
public:

```

```

#ifndef TO_CLEAN
    int reference_counter() const { return _count != 0 ? _count->_n : -1; }
#endif // TO_CLEAN
    counter* get_count() const { return _count; }
};

```

6.28 stack_allocator - stack-based allocator

(Source file: 'util/lib/stack_allocator.h')

Description

Stack-based allocator, conform to the STL specification of allocators. Designed to use stack-based data passed as a parameter to the allocator constructor. Does not "free" the memory. Assumes that if the allocator is copied, stack memory is cleared and new allocations begin at the bottom of the stack again.

Also works with any memory buffer, including heap memory. If the caller passes in heap memory, the caller is responsible for freeing the memory.

This allocator handles a limited area of memory: if this limit is reached, a "std::bad_alloc" exception is emitted. For a non-limited memory handler in the same spirit, see "heap_allocator"(9).

Example

```

const size_t stack_size = 1024;
vector<unsigned char> stack (stack_size);
stack_allocator<double> stack_alloc (stack.begin().operator->(), stack.size());
typedef map <size_t, double, less<size_t>, stack_allocator<pair<size_t,double> > >
map_type a (less<size_t>(), stack_alloc);
a.insert (make_pair (0, 3.14));
a.insert (make_pair (1, 1.17));
for (map_type::iterator iter = a.begin(), last = a.end(); iter != last; iter++)
    cout << (*iter).first << " " << (*iter).second << endl;
}

```

Implementation

```

template <typename T>
class stack_allocator {
protected:
    struct handler_type; // forward declaration:
public:

    // typedefs:

    typedef size_t          size_type;
    typedef std::ptrdiff_t  difference_type;
    typedef T*              pointer;

```

```

typedef const T*      const_pointer;
typedef T&            reference;
typedef const T&      const_reference;
typedef T            value_type;

// constructors:

stack_allocator() throw()
    : handler (new handler_type)
{
}
stack_allocator (unsigned char* stack, size_t stack_size) throw()
    : handler (new handler_type (stack, stack_size))
{
    trace_macro ("stack_allocator ctor");
}
stack_allocator (const stack_allocator& sa) throw()
    : handler (sa.handler)
{
    ++handler->reference_count;
}
template <typename U>
stack_allocator (const stack_allocator<U>& sa) throw()
    : handler ((typename stack_allocator<T>::handler_type*)(sa.handler))
{
    ++handler->reference_count;
}
~stack_allocator() throw()
{
    trace_macro ("stack_allocator dtor");
    check_macro (handler != NULL, "unexpected null mem_info");
    if (--handler->reference_count == 0) delete handler;
}
// Rebind to allocators of other types
template <typename U>
struct rebind {
    typedef stack_allocator<U> other;
};

// assignment:

stack_allocator& operator= (const stack_allocator& sa)
{
    handler = sa.handler;
    ++handler->reference_count;
    return *this;
}

// utility functions:

```

```

pointer      address (reference r)      const { return &r; }
const_pointer address (const_reference c) const { return &c; }
size_type    max_size() const { return std::numeric_limits<size_t>::max() / si

// in-place construction/destruction

void construct (pointer p, const_reference c)
{
    // placement new operator:
    new( reinterpret_cast<void*>(p) ) T(c);
}
// C++ 2011: default construct a value of type T at the location referenced by p
void construct (pointer p) { new ( reinterpret_cast<void*>(p) ) T(); }

void destroy (pointer p)
{
    // call destructor directly:
    (p)->~T();
}

// allocate raw memory

pointer allocate (size_type n, const void* = NULL)
{
    trace_macro ("allocate "<<n<<" type " << typename_macro(T));
    check_macro (handler->stack != NULL, "unexpected null stack");
    void* p = handler->stack + handler->allocated_size;
    handler->allocated_size += n*sizeof(T);

    if (handler->allocated_size + 1 > handler->max_size) {
        trace_macro ("stack is full: throwing...");
        throw std::bad_alloc();
    }
    return pointer (p);
}
void deallocate (pointer p, size_type n)
{
    trace_macro ("deallocate "<<n<<" type "<<typename_macro(T));
    // No need to free stack memory
}
const handler_type* get_handler() const {
    return handler;
}

// data:

protected:
    struct handler_type {

```

```

    unsigned char* stack;
    size_t         allocated_size;
    size_t         max_size;
    size_t         reference_count;

    handler_type()
        : stack (NULL),
          allocated_size (0),
          max_size (0),
          reference_count (1)
    {
        trace_macro ("stack_allocator::mem_info cstor NULL");
    }
    handler_type (unsigned char* stack1, size_t size1)
        : stack (stack1),
          allocated_size (0),
          max_size (size1),
          reference_count (1)
    {
        trace_macro ("stack_allocator::mem_info cstor1: size=" << max_size);
    }
    ~handler_type()
    {
        trace_macro ("stack_allocator::mem_info dstor: size=" << max_size);
    }
};
handler_type* handler;
template <typename U> friend class stack_allocator;
};
// Comparison
template <typename T1>
bool operator==( const stack_allocator<T1>& lhs, const stack_allocator<T1>& rhs) th
{
    return lhs.get_handler() == rhs.get_handler();
}
template <typename T1>
bool operator!=( const stack_allocator<T1>& lhs, const stack_allocator<T1>& rhs) th
{
    return lhs.get_handler() != rhs.get_handler();
}

```

7 Internal algorithms

7.1 iorheo - input and output functions and manipulation

(Source file: 'util/lib/iorheo.h')

Small pieces of code

input geo in standard file format:

```
cin >> g;
```

output geo in standard file format:

```
cout << g;
```

output geo in gnuplot format:

```
cout << gnuplot << g;
```

Description

output manipulators enable the selection of some pretty graphic options, in an elegant fashion.

Boolean manipulators

The boolean manipulators set an internal optional flag. Each option has its negative counterpart, as **verbose** and **noverbose**, by adding the **no** prefix.

```
cout << noverbose << a;
```

verbose trace some details, such as loading, storing or unix command operations on **cerr**. Default is on.

clean delete temporary files during graphic outputs. Default is on.

execute run unix operations, such as **gnuplot** or **plotmtv** or **vtk**. Note that the corresponding files are created. Default is on.

transpose perform transposition when loading/storing a **csr** matrix from Harwell-Boeing file. This feature is available, since the file format store matrix in transposed format. Default is off.

logscale when using matrix sparse postscript plot manipulator **ps** and **color**. The color scale is related to a logarithmic scale.

fill	
grid	
shrink	
tube	
ball	
full	
stereo	
cut	
iso	
split	when using the <code>vtk</code> or <code>mayavi</code> manipulators for a mesh or a field.
volume	volume rendering by using ray cast functions.
velocity	
deformation	Vector-valued fields are rendered by using arrows (<code>velocity</code>) or deformed meshes (<code>deformation</code>). For <code>vtk</code> or <code>plotmtv</code> rendering.
elevation	Scalar valued fields in two dimension are rendered by using a tridimensionnal surface elevation. For <code>vtk</code> or <code>plotmtv</code> rendering.
fastfieldload	try to reuse the supplied space. Default is on.

File format manipulators

The *format* manipulator group applies for streams. Its value is an enumerated type, containing the following possibilities:

<code>rheo</code>	use the default textual input/output file format. For instance, this is ' <code>.geo</code> ' for meshes, ' <code>.field</code> ' for discretized functions. This default format is specified in the corresponding class documentation (see also <code><undefined></code> [geo class], page <code><undefined></code> and <code><undefined></code> [field class], page <code><undefined></code>). This is the default.
<code>bamg</code>	uses ' <code>.bamg</code> ' Frederic Hecht's bidimensional anisotropic mesh generator file format for geo input/output operation.
<code>tetgen</code>	uses ' <code>.node</code> ' ' <code>.ele</code> ' and ' <code>.face</code> ' Hang Si's tridimensional mesh generator file format for geo input/output operation.
<code>mmg3d</code>	uses ' <code>.mmg3d</code> ' Cecile Dobrzynski's tridimensional anisotropic mesh generator file format for geo input/output operation.
<code>gmsh</code>	uses ' <code>.gmsh</code> ' gmsh Christophe Geuzaine and Jean-Francois Remacle mesh generator file format for geo input/output operation.
<code>gmsh_pos</code>	uses ' <code>.gmsh_pos</code> ' gmsh Christophe Geuzaine and Jean-Francois Remacle mesh metric file format for geo adapt input/output operation.
<code>grummp</code>	uses ' <code>.m</code> ' (bidimensional) or ' <code>.v</code> ' (tridimensionnal) Carl Ollivier-Gooch's mesh generator file format for geo input/output operation.

qmg	uses ' qmg ' Stephen A. Vavasis's mesh generator file format for geo input/output operation.
vtkdata	uses ' vtk ' mesh file format for geo input/output operations. This file format is suitable for graphic treatment.
vtkpolydata	uses ' vtk ' polydata (specific for polygonal boundaries) mesh file format for geo input/output operations. This file format is suitable for graphic treatment.
cemagref	uses ' cemagref ' surface mesh (topography, with a z cote). This file format is used at Cemagref (french research center for mountains, http://www.cemagref.fr).
dump	output an extensive listing of the class data structure. This option is used for debugging purpose.
hb	uses ' hb ' Harwell-Boeing file format for sparse matrix input/output operation. This is the default.
matrix_market	uses ' mm ' Matrix-Market file format for sparse matrix input/output operation.
ml	
matlab	uses ' m ' Matlab file format for sparse matrix output operation.
sparse_matlab	uses ' m ' Matlab sparse file format for sparse matrix output operation.
ps	uses ' ps ' postscript for sparse matrix output operation.
vtk	for mesh and field outputs. Generate ' vtk ' data file and the ' tcl ' command script file and run the vtk command on the ' tcl '.
mayavi	for field outputs. Generate ' vtk ' data file and the ' py ' command script file and run the python command on the ' py ' associated to the mayavi/vtk library.
geomview	for boundary cad outputs. Generate ' off ' data file and run the geomview command.
gnuplot	for mesh and field outputs. Generate ' gdat ' data file and the ' plot ' command script file and run the gnuplot command on the ' plot '.
plotmtv	for mesh and field outputs. Generate ' mtv ' data file and run the plotmtv command.
x3d	for mesh output. Generate ' x3d ' data file and run the x3d command. This tool has fast rotation rendering.
atom	for mesh output. Generate ' atom ' data file and run the PlotM command. Tridimensional mesh rendering is performed as a chemical molecule: nodes as balls and edges as tubes. The PlotM tool is developed at Cornell University Laboratory of Atomic and Solid State Physics (LASSP) in a Joint Study with IBM, with support by the Materials Science Center and Corning Glassworks.

Color manipulators

The *color* manipulator group acts for sparse matrix postscript output. Its value is an enumerated type, containing three possibilities:

```
cout << color << a;
cout << gray  << b;
cout << black_and_white << c;
```

The default is to generate a color postscript file. Conversely, its act for field rendering, via *mayavi*.

Valuated manipulators

Some manipulators takes an argument that specifies a value.

```
cout << geomview << bezieradapt << subdivide(5) << my_cad_boundary;
cout << vtk << iso << isovalue(2.5) << my_mesh;
cout << velocity << plotmtv << vectorscale(0.1) << uh;
```

See also [\[catchmark algorithm\]](#), page [\[undefined\]](#) for input-output of vector-valued fields.

isovalue float

n_isovalue int

n_isovalue_negative int

vectorscale float

subdivide float

rounding_precision float

This manipulator set a rounding precision that could be used by some output functions, e.g. by the *geo* class. By default, the rounding value is zero and there is no rounding.

image_format string

The argument is any valid image format, such as *png*, *jpg* or *pdf*, that could be handled by the corresponding graphic render.

7.2 typename_macro, pretty_typename_macro - type demangler and pretty printer

(Source file: 'util/lib/pretty_name.h')

Description

These preprocessor macro-definitions are useful when dealing with complex types as generated by imbricted template technics: they print in clear a complex type at run-time. *typeid_name_macro* obtains a human readable type in a *std::string* form by calling the system *typeid* function and then a demangler. When this type is very long, *pretty_name_macro* prints also it in a multi-line form with a pretty indentation.

Examples

```
typedef map <size_t, double, less<size_t>, heap_allocator<pair<size_t,double> > >
cout << typeid_name_macro (map_type);
```

Implementation

```
extern std::string typeid_name (const char* name, bool do_indent);
} // namespace rheolef

/// @brief get string from a type, with an optional pretty-printing for complex types
#define          typename_macro(T) rheolef::typeid_name(typeid(T).name(), false)
#define pretty_typename_macro(T) rheolef::typeid_name(typeid(T).name(), true)

/// @brief get string type from a variable or expression
template <class T> std::string          typename_of (T x) { return          typename_macro(x); }
template <class T> std::string pretty_typename_of (T x) { return pretty_typename_macro(x); }
```


8 Internal others

8.1 acinclude – autoconf macros

(Source file: ‘`config/acinclude.m4`’)

Description

These macros test for particular system features that rheolef uses. These tests print the messages telling the user which feature they are looking for and what they find. They cache their results for future `configure` runs. Some of these macros set some shell variable, defines some output variables for the ‘`config.h`’ header, or performs Makefile macros substitutions. See `autoconf` documentation for how to use such variables.

Synopsis

Follows a list of particular check required for a successful installation.

RHEO_CHECK_GINAC

Check to see if GiNaC library exists. If so, set the shell variable `rheo_have_ginac` to "yes", defines `HAVE_GINAC` and substitutes `INCLUDES_GINAC` and `LADD_GINAC` for adding in `CFLAGS` and `LIBS`, respectively, If not, set the shell variable `rheo_have_ginac` to "no".

RHEO_CHECK_CLN

Check to see if library `-lcln` exists. If so, set the shell variable `rheo_have_cln` to "yes", defines `HAVE_CLN` and substitutes `INCLUDES_CLN` and `LADD_CLN` for adding in `CFLAGS` and `LIBS`, respectively, If not, set the shell variable no "no". Includes and libraries path are searched from a given shell variable `rheo_dir_cln`. This shell variable could be set for instance by an appropriate `--with-cln=value.dir.cln` option. The default value is `/usr/local/math`.

RHEO_CHECK_SPOOLES_2_0

Check to see if spooles library has old version 2.0 since `FrontMtx_factorInpMtx` profile has changed in version 2.2. If so, defines `HAVE_SPOOLES_2_0`. This macro is called by `RHEO_CHECK_SPOOLES`.

RHEO_CHECK_TAUCS

Check to see if taucs library and headers exists. If so, set the shell variable `"rheo_have_taucs"` to "yes", defines `HAVE_TAUCS` and substitutes `INCLUDES_TAUCS` and `LADD_TAUCS` for adding in `CXXFLAGS` and `LIBS`, respectively, If not, set the shell variable to "no". Includes and libraries options are given shell variable `$rheo_ldadd_taucs` and `$rheo_incdire_taucs`. These shell variables could be set for instance by appropriate `"-with-taucs-ldadd='rheo_ldadd_taucs'` and `"-with-taucs-includes='rheo_incdire_taucs'` options.

RHEO_CHECK_BOOST_UBLAS

Check to see if boost headers exists. If so, set the shell variable `"rheo_have_boost"` to "yes", defines `HAVE_BOOST` and substitutes `INCLUDES_BOOST` for adding in `CXXFLAGS`, and `LDADD_BOOST` for adding in `LIBS`. If not, set the shell variable to "no". Includes options

are given in the shell variables `$rheo_incdir_boost` and `$rheo_libdir_boost`. These shell variables could be set for instance by appropriates `"--with-boost-includes='rheo_incdir_boost'"` and `"--with-boost-libdir='rheo_libdir_boost'"` options.

RHEO_CHECK_ARMADILLO

Check to see if armadillo headers exists. If so, set the shell variable `"rheo_have_armadillo"` to `"yes"`, defines `HAVE_ARMADILLO` and substitutes `INCLUDES_ARMADILLO` for adding in `CXXFLAGS`, and `LDADD_ARMADILLO` for adding in `LIBS`. If not, set the shell variable to `"no"`. Includes options are given in the shell variables `$rheo_incdir_armadillo` and `$rheo_libdir_armadillo`. These shell variables could be set for instance by appropriates `"--with-armadillo-includes='rheo_incdir_armadillo'"` and `"--with-armadillo-libdir='rheo_libdirarmadillo'"` options.

RHEO_CHECK_ZLIB

Check to see if zlib library and headers exists. If so, set the shell variable `"rheo_have_zlib"` to `"yes"`, defines `HAVE_ZLIB` and substitutes `INCLUDES_ZLIB` and `LADD_ZLIB` for adding in `CXXFLAGS` and `LIBS`, respectively, If not, set the shell variable to `"no"`. Includes and libraries path are searched from given shell variable `$rheo_dir_zlib/lib` and `$rheo_incdir_zlib`. Default value for `$rheo_incdir_zlib` is `$rheo_dir_zlib/include`. These shell variables could be set for instance by appropriates `"--with-zlib='dir_zlib'"` and `"--with-zlib-includes='incdir_zlib'"` options.

RHEO_CHECK_SPOOLES

Check to see if spooles library and headers exists. If so, set the shell variable `"rheo_have_spooles"` to `"yes"`, defines `HAVE_SPOOLES` and substitutes `INCLUDES_SPOOLES` and `LADD_SPOOLES` for adding in `CXXFLAGS` and `LIBS`, respectively, If not, set the shell variable to `"no"`. Includes and libraries path are searched from given shell variable `"rheo_libdir_spooles"` and `"rheo_incdir_spooles"`. These shell variables could be set for instance by appropriates `"--with-spooles='libdir_spooles'"` and `"--with-spooles-includes='incdir_spooles'"` options.

RHEO_CHECK_CHOLMOD

Check to see if cholmod library and headers exists. If so, set the shell variable `"rheo_have_cholmod"` to `"yes"`, defines `HAVE_CHOLMOD` and substitutes `INCLUDES_CHOLMOD` and `LADD_CHOLMOD` for adding in `CXXFLAGS` and `LIBS`, respectively, If not, set the shell variable to `"no"`. Includes and libraries path are searched from given shell variable `"rheo_libdir_cholmod"` and `"rheo_incdir_cholmod"`. These shell variables could be set for instance by appropriates `"--with-cholmod='libdir_cholmod'"` and `"--with-cholmod-includes='incdir_cholmod'"` options.

RHEO_CHECK_UMFPACK

Check to see if umfpack library and headers exists. If so, set the shell variable `"rheo_have_umfpack"` to `"yes"`, defines `HAVE_UMFPACK` and substitutes `INCLUDES_UMFPACK` and `LADD_UMFPACK` for adding in `CXXFLAGS` and `LIBS`, respectively, If not, set the shell variable to `"no"`. Includes and libraries path are searched from given shell variable `"rheo_libdir_umfpack"` and `"rheo_incdir_umfpack"`. These shell variables could be set for instance by appropriates `"--with-umfpack='libdir_umfpack'"` and `"--with-umfpack-includes='incdir_umfpack'"` options.

RHEO_CHECK_MALLOC_DBG

Check to see if malloc debug library `-lmalloc_dbg` and corresponding header `<malloc_dbg.h>` exists. If so, set the shell variable `rheo_have_malloc_dbg` to `"yes"`, defines `HAVE_MALLOC_DBG`,

add `-Idir_malloc_dbg/include` to CFLAGS, add `dir_malloc_dbg/lib/libmalloc_dbg.a` to LIBS. Here, `dir_malloc_dbg` is the directory such that `dir_malloc_dbg/bin` appears in PATH and the command `dir_malloc_dbg/bin/malloc_dbg` exists. If not, set the variable to "no". Set also LIBS_MALLOC_DBG to these flags.

RHEO_CHECK_DMALLOC

Check whether the dmalloc package exists and set the corresponding shell value "rheo_have_dmalloc" and HAVE_DMALLOC (in Makefile.am and config.h) accordingly, create LDADD_DMALLOC and LDADD_DMALLOCXX Makefile.am variables.

RHEO_CHECK_NAMESPACE

Check whether the namespace feature is supported by the C++ compiler. value. So, try to compile the following code:

```
namespace computers {
    struct keyboard { int getkey() const { return 0; } };
}
namespace music {
    struct keyboard { void playNote(int note); };
}
namespace music {
    void keyboard::playNote(int note) { }
}
using namespace computers;
int main() {
    keyboard x;
    int z = x.getkey();
    music::keyboard y;
    y.playNote(z);
    return 0;
}
```

If it compile, set the corresponding shell variable "rheo_have_namespace" to "yes" and defines HAVE_NAMESPACE. If not, set the variable no "no".

RHEO_CHECK_STD_NAMESPACE

Some compilers (e.g. GNU C++ 2.7.2) does not support the full namespace feature. Nevertheless, they support the "std:" namespace for the C++ library. is supported by the C++ compiler. The following code is submitted to the compiler:

```
#include<vector.h>
extern "C" void exit(int);
int main() {
    std::vector<int> x(3);
    return 0;
}
```

If it compile, set the corresponding shell variable "rheo_have_std_namespace" to "yes" and defines HAVE_STD_NAMESPACE. If not, set the variable no "no".

RHEO_PROG_GNU_MAKE

Find command make and check whether make is GNU make. If so, set the corresponding shell variable "rheo_prog_gnu_make" to "yes" and substitutes no_print_directory_option to "--no-print-directory". If not, set the shell variable no "no".

RHEO_CHECK_ISTREAM_RDBUF

RHEO_CHECK_IOS_BP

Check to see if "iostream::rdbuf(void*)" function exists, that set the "ios" buffer of a stream. Despite this function is standard, numerous compilers does not furnish it. a common implementation is to set directly the buffer variable. For instance, the CRAY C++ compiler implements this variable as "ios::bp". These two functions set the shell variables "rheo_have_istream_rdbuf" and "rheo_have_ios_bp" and define HAVE_ISTREAM_RDBUF and HAVE_IOS_BP respectively.

RHEO_CHECK_IOS_SETSTATE

Check to see if "ios::setstate(long)" function exists, that set the "ios" state variable of a stream. Despite this function is standard, numerous compilers does not furnish it. a common implementation is to set directly the buffer variable. For instance, the CRAY C++ compiler does not implements it. This function set the shell variables "rheo_have_ios_setstate" and define HAVE_IOS_SETSTATE.

RHEO_CHECK_FILEBUF_INT

RHEO_CHECK_FILEBUF_FILE

RHEO_CHECK_FILEBUF_FILE_MODE

Check whether "filebuf::filebuf(int fileno)", "filebuf::filebuf(FILE* fd)" exist, or "filebuf::filebuf(FILE* fd, ios::openmode)" exist, respectively. If so, set the corresponding shell variable "rheo_have_filebuf_int" (resp. "rheo_have_filebuf_file") to "yes" and defines HAVE_FILEBUF_INT, (resp. HAVE_FILEBUF_FILE). If not, set the variable no "no". Notes that there is no standardisation of this function in the "c++" library. Nevertheless, this functionality is useful to open a pipe stream class, as "pstream(3)".

RHEO_CXX_CLOCK_GETTIME

Tries to determine whether clock_gettime is useable.

RHEO_CHECK_GETTIMEOFDAY

Check whether the "gettimeofday(timeval*, timezone*)" function exists and set the corresponding shell value "rheo_have_gettimeofday" and define HAVE_GETTIMEOFDAY accordingly.

RHEO_CHECK_WIERDGETTIMEOFDAY

This is for Solaris, where they decided to change the CALLING SEQUENCE OF gettimeofday! Check whether the "gettimeofday(timeval*)" function exists and set the corresponding shell value "rheo_have_wierdgettimeofday" and define HAVE_WIERDGETTIMEOFDAY accordingly.

RHEO_CHECK_BSDGETTIMEOFDAY

For BSD systems, check whether the "BSDgettimeofday(timeval*, timezone*)" function exists and set the corresponding shell value "rheo_have_bsdgettimeofday" and define HAVE_BSDGETTIMEOFDAY accordingly.

RHEO_CHECK_AMICCLK

Check whether the clock "amicclk()" function exists and set the corresponding shell value "rheo_have_amicclk" and define HAVE_AMICCLK accordingly.

RHEO_CHECK_TEMPLATE_FULL_SPECIALIZATION

Check whether the template specialization syntax "template<>" is supported by the compiler value. So, try to compile, run and check the return value for the following code:

```
template<class T> struct toto {
    int tutu() const { return 1; }
};
template<> struct toto<float> {
    int tutu() const { return 0; }
};
main() {
    toto<float> x;
    return x.tutu();
}
```

If so, set the corresponding shell variable "rheo_have_template_full_specialization" to "yes" and defines HAVE_TEMPLATE_FULL_SPECIALIZATION. If not, set the variable no "no".

RHEO_CHECK_ISNAN_DOUBLE

RHEO_CHECK_ISINF_DOUBLE

RHEO_CHECK_FINITE_DOUBLE

RHEO_CHECK_INFINITY

RHEO_CHECK_ABS_DOUBLE

RHEO_CHECK_SQR_DOUBLE

Check whether the functions

```
bool isnan(double);
bool isinf(double);
bool finite(double);
double infinity();
double abs();
double sqr();
```

are supported by the compiler, respectively. If so, set the corresponding shell variable "rheo_have_XXX" to "yes" and defines HAVE_XXX. If not, set the variable no "no".

RHEO_CHECK_FLEX

Check to see if the "flex" command and the corresponding header "FlexLexer.h" are available. If so, set the shell variable "rheo_have_flex" to "yes" and substitutes FLEX to "flex" and FLEXLEXER_H to the full path for FlexLexer.h If not, set the shell variable no "no".

RHEO_PROG_CC_INTEL

Check whether we are using INTEL C++ compiler. If so, set the shell variable "ac_cv_prog_icc" to "yes" and define HAVE_INTEL_CXX. If not, set the shell variable no "no". The shell variable is also exported for sub-shells, such as ltconfig from libtool.

RHEO_PROG_CC_CLANG

Check whether we are using CLANG C++ compiler. If so, set the shell variable "ac_cv_prog_clang" to "yes" and define HAVE_INTEL_CLANG. If not, set the shell variable no "no". The shell variable is also exported for sub-shells, such as ltconfig from libtool.

RHEO_CLANG_LINK_LIBSTDCXX

Check whether CLANG C++ compiler links correctly with std c++ lib. If not, exit with error. This is still a bug in debian/testing (stretch) in june 2016. https://llvm.org/bugs/show_bug.cgi?id=24844

RHEO_RECOGNIZE_CXX

Check whether we are able to recognize the C++ compiler. Tested compilers:

```
The GNU    C++ compiler that defines: __GNUC__   (egcs-1.1.1)
The INTEL  C++ compiler that defines: __ICC      (ICPC-12)
```

If so, substitute RECOGNIZED_CXX to a specific compiler's rule file, e.g, "\${top_srcdir}/config/gnu.cxx.mk" for a subsequent Makefile include. If not, substitute to "/dev/null". Substitutes also EXTRA_LDFLAGS. Raw cc is the C compiler associated to the C++ one. By this way C and C++ files are handled with a .c suffix. Special C files that require the cc compiler, such as "alloca.c" use some specific makefile rule.

usage example:

```
AC_PROG_CC(gcc cc icc cl)
AC_PROG_CXX(c++ g++ cxx icpc KCC CC CC cc++ xlc aCC)
RHEO_RECOGNIZE_CXX
```

RHEO_GXX2011_PRE

Check for the "-std=c++0x" support for g++. Requires a recent version of the GNU C++ compiler (>= 4.5).

RHEO_GXX2011

Check for the "-std=c++11" support for g++. Requires a recent version of the GNU C++ compiler (>= 4.7).

RHEO_OPTIMIZE_CXX

Set some optimization flags associated to the recognized C++ compiler and platform.

RHEO_CHECK_LATEX_HYPERREF

Check whether the hyperref LaTeX package exists and set the corresponding shell value "rheo_have_latex_hyperref" and HAVE_LATEX_HYPERREF (for Makefiles) accordingly.

RHEO_CHECK_MPI

Check for the "mpirun" command, the corresponding header "mpi.h" and library "-lmpi" are available. If so, set the shell variable "rheo_have_mpi" to "yes", and substitutes MPIRUN to "mpirun" and RUN to "mpirun -np 2", INCLUDES_MPI and LDADD_MPI. If not, set the shell variable no "no".

RHEO_CHECK_METIS

Check for the "cgall" computational geometry library. Defines HAVE_METIS and substitutes INCLUDES_METIS and LDADD_METIS.

RHEO_CHECK_PARMETIS

Check for the "parmetis" libraries. Defines HAVE_PARMETIS and substitutes INCLUDES_PARMETIS and LDADD_PARMETIS. Requires the MPI library.

RHEO_CHECK_SCOTCH

Check for the "scotch" distributed mesh partitioner libraries. Defines HAVE_SCOTCH and substitutes INCLUDES_SCOTCH and LDADD_SCOTCH. Requires the MPI library.

RHEO_CHECK_BOOST

Check for the "boost" library. Defines HAVE_BOOST and substitutes LDADD_BOOST and INCLUDES_BOOST.

RHEO_CHECK_BLAS

Check for the "blas" basic linear algebra subroutines library. Defines HAVE_BLAS and substitutes LDADD_BLAS and INCLUDES_BLAS.

RHEO_CHECK_SCALAPACK

Check for the "scalapack" basic linear algebra subroutines library. Defines HAVE_SCALAPACK and substitutes LDADD_SCALAPACK and INCLUDES_SCALAPACK.

RHEO_CHECK_TRILINOS

Check for the "trilinos" distributed preconditioner libraries. Defines HAVE_TRILINOS and substitutes INCLUDES_TRILINOS and LDADD_TRILINOS. Requires the MPI and SCOTCH libraries.

RHEO_CHECK_PASTIX

Check for the "pastix" sequential or distributed direct solver libraries, depending on the "rheo_use_distributed" shell variable. Defines HAVE_PASTIX and substitutes INCLUDES_PASTIX and LDADD_PASTIX.

RHEO_CHECK_MUMPS

Check for the "mumps" distributed direct solver libraries. Defines HAVE_MUMPS and substitutes INCLUDES_MUMPS and LDADD_MUMPS. Requires the MPI and SCOTCH libraries.

RHEO_CHECK_MUMPS_WITH_PTSCOTCH_SUPPORT

Check whether "mumps" is configured with the "ptscotch" support Defines HAVE_MUMPS_WITH_PTSCOTCH and the shell variable rheo_have_mumps_with_ptscotch.

RHEO_CHECK_MUMPS_WITH_SCOTCH_SUPPORT

Check whether "mumps" is configured with the "scotch" support Defines HAVE_MUMPS_WITH_SCOTCH and the shell variable rheo_have_mumps_with_scotch.

RHEO_CHECK_MUMPS_WITH_PARMETIS_SUPPORT

Check whether "mumps" is configured with the "parmetis" support Defines HAVE_MUMPS_WITH_PARMETIS and the shell variable rheo_have_mumps_with_parmetis.

RHEO_CHECK_MUMPS_WITH_METIS_SUPPORT

Check whether "mumps" is configured with the "metis" support Defines HAVE_MUMPS_WITH_METIS and the shell variable rheo_have_mumps_with_metis.

RHEO_CHECK_STD_INITIALIZER_LIST

Some compilers (e.g. GNU C++ 4.4.x) does not support the std::initializer_list feature. Set the corresponding shell variable "rheo_have_std_initializer_list" to "yes" and defines HAVE_STD_INITIALIZER_LIST. If not, set the variable no "no".

RHEO_CHECK_GMP

Check for the "gmp" gnu multi-precision library. Defines HAVE_GMP and substitutes INCLUDES_GMP and LDADD_GMP.

RHEO_CHECK_MPFR

Check for the "mpfr" gnu multi-precision library. Defines HAVE_MPFR and substitutes INCLUDES_MPFR and LDADD_MPFR.

RHEO_CHECK_CGAL

Check for the "cgall" computational geometry library. Defines HAVE_CGAL and substitutes INCLUDES_CGAL and LDADD_CGAL.

RHEO_DEBIAN_FIX_LIBTOOL

Fix rpath libtool issue for debian packaging. See also <http://wiki.debian.org/RpathIssue>

RHEO_CHECK_FLOAT128

Check for the "float128" arithmetic (boost multiprecision and GNU quadmath). Defines HAVE_FLOAT128 and substitutes INCLUDES_FLOAT128 and LDADD_FLOAT128.

9 FAQ for developers

This list of Frequently Asked Questions intended for Rheolef developers and maintainers. I'm looking for new questions (*with* answers, better answers, or both. Please, send suggestions to Pierre.Saramito@imag.fr.

9.1 How to regenerate the configure script

The configure script and makefiles are automatically produced from file 'configure.ac' and 'Makefile.am' by using the autoconf and automake commands. Enter:

```
bootstrap
```

9.1.1 In which order does the things build ?

Let us look at with details the configure files flow:

```
[acinclude.m4] -----> aclocal* -----> [aclocal.m4]

[configure.ac] -+
-----+----> autoconf* -----> configure
[aclocal.m4] ---+

[Makefile.am] -----> automake* -----> Makefile.in

[config.h.in] -+                               +-> config.h
               |                               |
Makefile.in ---+----> configure* -----+--> Makefile
               |                               |
[config.mk.in] +                               +-> config.mk
```

9.1.2 What means these commands ?

Let us review the list of commands:

aclocal take 'acinclude.m4' and build 'aclocal.m4'. The arguments specifies that these files are located in the 'config/' directory.

automake translate every 'Makefile.am' and then build a 'Makefile.in'.

autoconf translate 'configure.ac' in 'configure' which is an executable shell script. The arguments specifies that 'aclocal.m4' is located in the 'config/' dirctory. All this files are machine independent.

configure

the automatically generated script, scan the machine and translate 'config.h.in', 'config.mk.in' and every 'Makefile.in' into 'config.h', 'config.mk' and every 'Makefile', respectively. At this step, all produced files are machine dependent.

9.2 How to save my version ?

First, check that our distribution is valid

```
make distcheck
```

First, check that your modifications are not in conflict with others. Go to the top source directory and enter:

```
make status
```

9.2.1 Easy: no conflicts with another developer

A listing of labeled files appears:

```
Modified      skit/lib/blas1_tst.c
Update        skit/lib/blas2_tst.c
```

It means that you have modified 'blas1_tst.c'. Another concurrent developer has modified 'blas2_tst.c', and your local file version is not up-to-date. There is no conflict, labeled by a **Merge** label.

First, update your local version:

```
make update
```

Before to store your version of the Rheolef distribution, check the consistency by running non-regression tests:

```
make distcheck
```

When all tests are ok:

```
make save
```

and enter a change log comment terminated by the **ctrl-D** key.

Check now that your version status is the most up-to-date Rheolef distribution:

```
make status
```

9.2.2 I have conflicts with another developer

The listing issued by **make status** looks like:

```
Modified      skit/lib/blas1_tst.c
Update        skit/lib/blas2_tst.c
*Merge*       skit/lib/blas3_tst.c
```

It means that you and another developer have modified at least one common line in 'blas3_tst.c'. Moreover, the developer has already saved his version in a previous Rheolef distribution. You have now to merge these modifications. Thus, enter:

```
cd skit/lib
mv blas3_tst.c blas3_tst.c.new
cvs update blas3_tst.c
sdiff blas3_tst.c blas3_tst.c.new | more
```

and then edit 'blas3_tst.c' to integrate the modifications of 'blas3_tst.c.new', generated by another developer. When it is done, go to the top directory and enter,

```
make status
```

It prints new:

```
Modified          skit/lib/blas1_tst.c
Update            skit/lib/blas2_tst.c
Modified          skit/lib/blas3_tst.c
```

The situation becomes mature:

```
make update
```

It will update the 'blas2_tst.c'. Finally,

```
make save
```

that will save modified 'blas1_tst.c' and 'blas3_tst.c'.

9.2.3 I have deleted a source file...

I have entered:

```
rm Makefile.am
```

...aie !

How to restaure the file, now ?

Enter:

```
cvs update Makefile.am
```

You restaure the last available version of the missing file in the most up-to-date Rheolef distribution.

Appendix A The GNU General Public License

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <https://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.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

one line to give the program’s name and a brief idea of what it does.

Copyright (C) year name of author

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 <https://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

program Copyright (C) year name of author

This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.

This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <https://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <https://www.gnu.org/licenses/why-not-lgpl.html>.

Concept Index

A

animation 13, 52
axisymmetric coordinate system 34, 38
axisymmetric geometry 78

B

bugs 11

C

`cemagref` topographic mesh 203
configure 47, 127
conjugate gradien algorithm 125, 156
conjugate gradient algorithm 148, 154
continuation method 139
continuation methods 13, 52
corner singularity 138

D

debugging 7, 209
deformation 202
determinant 123
diagonal matrix 110, 150
direct solver 123

E

edge 169
elevation 14, 202
environment sanity check writes 48

F

FAQ 215
finite element 87
finite element method 125, 156

G

generalized minimum residual method 150
geometrical element 169
graphic render 16, 21, 22, 201, 204

H

hexaedron 175

I

image file format 16, 22, 204
incompressible elasticity 125, 156
installing 3, 47
integrate 147
iterative solver 123, 148, 150, 154, 157

L

Lagrange-Galerkin method 54
local matrix inversion 78

M

Makefile 47
mass lumping 78
mesh 28, 29, 31, 32, 35, 36, 38, 39
mesh boundary 161, 163, 164
mesh graphic representation 20
mesh order 30, 38
mesh subdividing 22
method of characteristic 54
mixed linear problem 125, 156
multifrontal linear solver 5, 9

N

Newton method 139, 141, 145
nonlinear problem 139, 141, 145
numbering, global degree of freedom 164

O

out-of-core sparse linear solver 9

P

plotting	15, 20, 26
plotting data	15
plotting mesh	20
point	177
polynomial basis	84, 164, 166, 167, 168
porting the code	207
preconditioner	148, 154, 157
preconditionner	124
prism	177
problems	11
projection	14, 18

Q

quadrangle	178
quadrature formula	146
quadrature formulae	54, 94, 180
quarature formula	65

R

reference counting	194
reference element	84, 166, 169, 175, 177, 178, 181, 183, 185
RHEOPATH environment variable	13, 16, 21, 119, 132
riesz representer	146

S

shallow copy	194
smart pointer	194
sparse matrix	5
stabilized mixed finite element method ...	125, 156
standard template library (STL)	6
stereo 3D rendering	21
Stokes problem	125, 156
stream function	18
supernodal symmetric linear solver	8

T

tensor	95
tensor3	98
tensor4	100
tetrahedron	183
time-dependent problems	13, 52
topography	14, 203
triangle	185

U

Uzawa algorithm	157
-----------------------	-----

V

velocity	202
version management	217
vorticity	18

Program Index

B

`bang` 31, 35
`bang2geo` 28
`basis` 26
`branch` 13

C

`configure` 3, 127

D

`dmalloc` 7, 209

F

`field` 14, 15

G

`geo` 20, 28, 29, 31, 32, 35, 36, 38
`gmsh` 30, 36, 38, 175, 183

L

`latex` 212

M

`mfield` 14, 204
`mkgeo_ball` 29
`mkgeo_contraction` 31
`mkgeo_grid` 32
`mkgeo_sector` 35
`mkgeo_ugrid` 36
`msh2geo` 175, 183

R

`rheolef-config` 4, 47

Class Index

A

ad3 83
 adapt_option 137

B

basis 84, 166, 167, 168
 basis_option 167

C

catchmark 16, 159, 204
 csr 123, 150, 201

D

dia 110, 150
 domain_indirect 161

F

field 51, 52, 95, 137, 143, 146, 159, 201

G

geo 20, 51, 54, 89, 137, 143, 169, 201
 geo_domain 163
 geo_domain_indirect_rep 164
 geo_element 169

I

idiststream 119
 iorheo 132, 201
 irheostream 132

N

numbering 164

O

odiststream 119
 orheostream 132

P

point 89, 95

Q

quadrature 94, 180

R

reference_element 169
 reference_element 84, 94, 166, 167, 168, 169,
 180, 181

S

smart_pointer 194
 smart_pointer_clone 194
 solver 123, 127
 space 54, 79, 87, 143, 146, 167, 168

T

tensor 95
 tensor3 98
 tensor4 100

V

vec 110, 123, 150
 Vector 190

Approximation Index

P0	14	P1	14
----------	----	----------	----

Function Index

A

adapt 137
 append_dir_to_rheo_path 132

C

cg 148, 156
 cg_abtb 156
 cg_abtbcb 156
 continuation 55, 139

D

damped_newton 141
 delete_suffix 132
 derr 119
 din 119
 dout 119
 dual 57

F

fastfieldload 202
 file_exists 132
 ftos 132

G

get_basename 132
 get_dirname 132
 get_full_name_from_rheo_path 132

H

has_suffix 132

I

integrate 78, 141, 147
 interpolate 56, 143
 itos 132

L

level_set 51, 143

M

minres 154, 156
 minres_abtb 156
 minres_abtbcb 156

N

newton 145
 normal 57

P

prepend_dir_to_rheo_path 132

Q

qmr 150

R

riesz 54, 146

S

scatch 132

U

uzawa 157

File Format Index

- - ‘.1’, ‘.3’,... unix manual pages 3
 - ‘.atom’ PlotM mesh 201, 202
 - ‘.bang’ bang mesh 28, 39, 202
 - ‘.bang’ mesh file 16, 21
 - ‘.bang_bb’ data file 16
 - ‘.bmp’ image 16, 22
 - ‘.branch’ family of fields 13
 - ‘.cemagref’ cemagref topographic mesh 203
 - ‘.dmn’ domain names 28
 - ‘.ele’ tetgen mesh elements 202
 - ‘.eps’ image 16, 22
 - ‘.face’ tetgen mesh boundary faces 202
 - ‘.field’ field 14, 15, 201, 202
 - ‘.gdat’ gnuplot data 201
 - ‘.geo’ mesh 28, 29, 31, 32, 35, 36, 38, 201, 202
 - ‘.gif’ image 16, 22
 - ‘.gmsh’ gmsh mesh 202
 - ‘.gmsh’ mesh file 16, 22
 - ‘.gmsh_pos’ data file 16
 - ‘.gmsh_pos’ gmsh metric mesh 202
 - ‘.gz’ gzip 119, 132
 - ‘.hb’ Harwell-Boeing matrix 201, 202
 - ‘.info’ GNU info 3
 - ‘.jpg’ image 16, 22, 204
 - ‘.m’ grummp bidimensionnal mesh 202
 - ‘.m’ matlab matrix 202
 - ‘.mesh’ mmg3d mesh 202
 - ‘.mm’ Matrix-Market matrix 201
 - ‘.msh’ gmsh mesh 38
 - ‘.mtv’ plotmtv 201, 202
 - ‘.node’ tetgen mesh nodes 202
 - ‘.off’ geomview data 201
 - ‘.pdf’ image 16, 22, 204
 - ‘.plot’ gnuplot script 201, 202
 - ‘.png’ image 16, 22, 204
 - ‘.ppm’ image 16, 22
 - ‘.ps’ image 16, 22
 - ‘.ps’ postscript 3, 202
 - ‘.py’ python script file (for mayavi visualization tool) 203
 - ‘.svg’ image 16, 22
 - ‘.tcl’ tool command language 201
 - ‘.tif’ image 16, 22
 - ‘.v’ grummp tridimensionnal mesh 202
 - ‘.vtk’ mesh file 21
 - ‘.vtk’ visualization toolkit 201, 202
 - ‘.x3d’ x3d mesh 201, 202
-
- ## A
- acinclude.m4 215
-
- ## C
- configure.ac 215
-
- ## M
- Makefile.am 215

Related Tool Index

A

aclocal 215
armadillo, basic linear algebra subroutines 4
armaidillo, generic dense/sparse matrix library .. 7
autoconf 207, 215
automake 215

B

bang 16, 21, 22, 25, 28, 39, 137, 202
bash 4
blas, basic linear algebra subroutines 8
boost, extensions of the c++ standard template
 library 4
boost, generic dense/sparse matrix library 7

C

cgal, computational geometry library 4, 9
cholmod, sequential choleski supernodal solver
 library 8
cln, arbitrary precision float library 7
cray c++ compiler 6
cray unicos, operating system 6
csh 4
cvs 217

D

debian 7
dmalloc, debug runtime library 7, 209

F

float: quadruple or extended floating arithmetic
 library 7

G

geomview 201
ghostview 3
ginac, not a computer algebra system 5
gmsh 5, 16, 22, 25, 38, 137
gnu c++ compiler 6
gnuplot 5, 13, 17, 21, 27, 201, 202
gzip 132

H

hpux, operating system 4, 6, 48

I

info 3
intel c++ compiler 6
irix, operating system 6

K

kai c++ compiler 6

M

mac osx, operating system 6
make 3, 217
Makefile 3, 7
man 3
mayavi 17, 21, 203
mayavi2 5
metis, mesh partitioner 5
mmg3d 202
mpi, message passing interface 5
msh2geo 38
mumps, distributed direct solver 8
mumps, multifrontal massively distributed sparse
 direct solver 5

P

paraview 5, 13, 17, 21
parmetis, distributed mesh partitioner 8
pastix, distributed direct solver 8
plotmtv 13, 201, 202
PlotM 201, 202

S

scalapack, scalable linear algebra package 8
scotch, distributed mesh partitioner 8
scotch, mesh partitioner 5
sh 4
spooles, multifrontal solver library 9

T

taucs, out-of-core sparse solver library	9
tetgen	202
trilinos, distributed incomplete choleski factorization	8
trilinos, large-scale object-oriented solvers framework	5

U

umfpack, multifrontal sparse direct solver	4
umfpack, sequential multifrontal solver library ..	9

V

vtk	13, 21, 201, 202
-----------	------------------

X

x3d	202
-----------	-----