Rheolef, the finite element system.

Reference manual
Version 5.93
6 March 2011

by Pierre Saramito and Jocelyn Etienne

1 Abstract

`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`.

2 Installing rheolef

(Source file: 'configure.ac')

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

Successfull compilation and installation require the GNU make command. Check your installation by running a small test:

```
make installcheck
```

2.1 Reading the documentation

rheolef comes with three documentations:

- an users guide with a set of complete examples
- 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. The users guide is generated in pdf format:

```
evince usrman.pdf
```

The reference manual is generated in html format:

```
firefox http://ljk.imag.fr/membres/Pierre.Saramito/rheolef/rheolef.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 3 field
```

The browsable source code is generated in html format:

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

2.2 Using and alternative installation directory

The default install prefix is '/usr/local'. If you prefer an alternative directory, e.g. 'HOME/sys', you should enter:

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

In that case, you have to add the folowing lines at the end of your '.profile' or '.bash_profile' file:

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

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

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

```
echo $SHELL
```

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

```
source ~/.cshrc
```

hpux users should replace `LD_LIBRARY_PATH` by `SHLIB_PATH`. If you are unure, 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
```

2.3 Recommanded library

The use of implicit numerical schemes leads to the resolution of large sparse linear systems. `rheolef` installation optionally recognizes both `umfpack` and `spooles`, two multifrontal direct solvers libraries, and `taucs`, an out-of-core sparse direct solver, When no efficient sparse direct solver library is not available, rheolef uses a direct solver based upon a traditionnal skyline Choleski factorisation combined with a reverse Cuthill-Mc Kee reordering. Here is the time complexity for 2d and 3d regular grid-based Poisson problems.

```
                    skyline     multifrontal
factorization       N^2         N log N
 2d
resolution         N^3/2        N log N

factorization       N^7/3       N^2
 3d
resolution         N^5/2        N^4/3
```

The out-of-core strategy is efficient for very large problems or if your computer has few memory. The complexity estimation has been shown by A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10:345-363, 1973.

2.4 The umfpack library

The `umfpack` library, if present, is auto-detected and used. It is widely available, as a debian package.

2.5 The spooles library

The optional `spooles-2.2` multifrontal linear solver from A. George, october 1998, is recommended for solving efficiently large linear systems.

It is available at `http://www.netlib.org/linalg/spooles/`. This library is free software. A default skyline strategy is used when it is not founded by `configure`. Both the version 2.2. and the older `2.0` library interface are supported by `rheolef`.

First build the spooles library:

```
cd /usr/local/src
mkdir spooles-2.2
gzip -dc < ../spooles.2.2.tar.gz | tar xvf -
make CC=gcc CFLAGS="-O9" lib
```

Better, you can build spooles as a shared library, as follow (assuming GNU C++):

```
make CC=gcc CFLAGS="-O9 -fPIC" lib
mkdir tmp
cd tmp/
ar x ../spooles.a
gcc -shared -o ../libspooles.so *.o
cd ..
rm -rf tmp spooles.a
make clean
```

Note that on `hpux` systems, the '`.sl`' suffix is used instead of the '`.so`' one. Then, go to the rheolef directory:

```
./configure --with-spooles=/usr/local/src/spooles-2.2
make
make install
```

2.6 The taucs library

The optional `taucs-2.0` out-of-core sparse linear solver from S. Toledo and D. Chen, may 2002, is recommended for very large linear systems or if your computer has few memory. `taucs` can factor a matrix whose factors are larger than the main memory by storing the factors on disk files. In that case, this strategy is significantly faster than paging activity when running an in-core algorithm such as `spooles`.

It is available at `http://www.tau.ac.il/~stoledo/taucs/`. This library is free software.

The `taucs` installation requires `metis`, `lapack` and `blas` libraries. Assuming all these libraries are installed in `/usr/local/lib` and the taucs header '`taucs.h`' is located in `/usr/local/include`, the configuration for `rheolef` writes:

```
./configure --with-taucs-ldadd='-ltaucs -llapack -lblas -lmetis -lg2c'
```

An alternative is to `blas` is to use the more efficient `atlas` libraries. See the installation notes for the taucs library.

2.7 Too long compilations

On platforms with small memory, the `c++` compilation could be long. By default, the `configure` scripts uses a `-O2` compilation level. Turn it off either by:

```
setenv CXXFLAGS "-O0"
./configure
make
```

or by editing directly '`config/config.mk`'.

2.8 Portability issues

rheolef is based on STL. Therefore you need a compiler that supports the ANSI C++ standards. STL and rheolef especially depend heavily on the template support of the compiler.

current version of rheolef has only been tested with

```
GNU  C++ compiler (g++-2.95.4, 2.96, 3.0.2, 3.0.4, 4.0, 4.3)
Compaq C++ compiler (V6.3, V6.5)
```

on the following operating systems:

```
linux debian, pentium 3 and 4, using GNU C++
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++
```

The GNU C++ compiler is a free software available at `http://www.gnu.org`. The Compacq C++ compiler is available on Compacq (DEC) platforms.

Previous 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:

```
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
linux (redhat, suze and slackware), pentium (2,3,pro)
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.

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

```
make check
```

and reports us the result. If all is ok, please, send us a mail, and we will add your configuration to the list, and else we will try to circumvent the problem.

2.9 Known portability limitations

On compacq alpha, the old cxx V6.1 (DIGITAL UNIX V4.0) compiles well but scratches on non-regression tests, due to a bug in shared libraries managment. Please, update to cxx V6.3 or more, it will work well.

On some hpux and GNU C++ version combinations, building shared libraries could be broken. Uses instead:

```
./configure --disable-shared
```

It should work. Building static libraries leads to larger executable files, and then more disk space available. This limitation depends upon compilers and systems compatibilities, not upon rheolef source code portability.

Please read `http://www.gnu.org/software/gcc/install/specific.html`. A recent version of `binutils` and a hpux patch are recommended.

Osman F. Buyukisik `<osman.buyukisik@ae.ge.com>` reported that

```
export LIBS=" -ldce -static "
./configure
```

works also well on hpux-10.20.

2.10 Related tools

Using rheolef suggests the use of the following optional tools:

```
gnuplot
plotmtv
mayavi
paraview
vtk
geomview
qmg
grummp
```

2.11 Advanced configuration

Note that each `--with` option has a corresponding `--without` option.

`--enable-optim`
: Turns compile-time optimization to maximum available. Default is on.

`--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`'.

`--with-doubledouble`
: Uses `doubledouble` class as the default rheolef `Float` type. This option is usefull for a quadruple-like precision on machines where this feature is not or incorrectly available. Default is off.

`--enable-debian-packaging=`*debian_packaging*
: Generate files the debian way by respecting the 'Debian Policy Manual'.

`--with-boost-incdir=`*incdir_boost*
: Turns on/off the `boost` boost/numeric/uBlas dense and sparse matrix 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-distributed`
: Turns on/off the distributed memory and parallel computation features. Default is off. This feature is currently in development. When on, configure try to detect either the mpi, and cotch or the parmetis libraries.

`--enable-mpi`
: Turns on/off the distributed memory and parallel 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`
`--with-scotch=`*libdir_scotch*
: Turns on/off the `scotch` distributed mesh partitionner. 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`
`--with-parmetis=`*libdir_parmetis*
: Turns on/off the `parmetis` distributed mesh partitionner. 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-ldadd`
`--with-blas-ldadd=`*rheo_ldadd_blas*
: Turns on/off the `blas` libraries. Check in *rheo_ldadd_blas* for `libblas.so`, or the corresponding `.a` libraries. Default is to auto-detect. This librariy is required for the pastix library.

`--with-pastix`
`--with-pastix=`*libdir_pastix*
: Turns on/off the `scotch` distributed mesh partitionner. 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-umfpack=`*libdir_umfpack*
`--with-umfpack-incdir=`*incdir_umfpack*
: Turns on/off the `umfpack` version 4.3 multifrontal direct solver. Default *incdir_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. 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-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 *incdir_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. Generic or hardware-dependant optimization

2.12 Future configure options

These options will be implemented in the future:

`--with-long-double`
: Defines `long double` as the default rheolef `Float` type. Default is off and defines `double` as `Float` type. Warning: most of architectures does not provides mathematical library in `long double`, e.g. `sqrt((long double)2)` returns only double precision result on Sun architectures, despites it uses a double memory space. Thus use `--with-doubledouble` instead.

2.13 Rebuilding the documentation

You can rebuild the documentation by entering:

```
make dvi
```

The user's manual goes into '`doc/usrman/`' and the the reference manual into '`doc/refman/`'.

The user's manual becomes available in '`.dvi`', '`.ps`' and '`.pdf`' formats.

Regenerating the full documention requires also optional tools.

```
xdvi      doc/usrman/usrman.dvi
ghostview doc/usrman/usrman.ps
xpdf      doc/usrman/usrman.pdf
```

The reference manual is available in '`.info`', '`.dvi`', '`.ps`', '`.pdf`', '`.html`' and '`.txt`' formats:

```
info -f   doc/refman/rheolef.info
xdvi      doc/refman/rheolef.dvi
ghostview doc/refman/rheolef.ps
xpdf      doc/refman/rheolef.pdf
lynx      doc/refman/rheolef_toc.html
vi        doc/refman/rheolef.txt
```

2.14 Documentation using doxygen

`doxygen` is a documentation system and can be found at `http://www.doxygen.org`.

`doxygen` has built-in support to generate inheritance diagrams for C++ classes. `doxygen` can use the `dot` tool from `graphviz` to generate more advanced diagrams and graphs. `graphviz` is an open-sourced, cross-platform graph drawing toolkit from AT&T and Lucent Bell Labs and can be found at `http://www.research.att.com/sw/tools/graphviz/`.

```
mozilla file:‘pwd‘/doc/doxygen/html/index.html
```

2.15 Compile-time optional tools

Installation has been tested with the indicated version number. Later version should also work well.

`doubledouble-2.2`
: The definitive site for this code is

  `http://www-epidem.plantsci.cam.ac.uk/~kbriggs/doubledouble.html`.

  The 2.2 version is included in this distribution for convenience.

`cln-1.0.2`
: Bruno Haible's arbitrary precision float library.

  See `http://clisp.cons.org/~haible/packages-cln.html`.

`dmalloc-4.8.2`
: For dynamic allocation checking in conjuction with `configure --with-debug`.

2.16 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, renumbering and factorization, linear solvers.
`fem`	finite element library, mesh, spaces and forms.
`doc`	reference manual, user's guide, web site.
`examples`	demonstrations as in users'guide.

2.17 Compiling from development version

The installation process is preceeded by an extra `bootstrap` stage, and the `configure` stage takes an extra `--enable-ginac` flag. An example writes:

```
./bootstrap
./configure --enable-ginac
make
make install
```

You will need some extra-development tools that are used by maintainers.

• for configuration files:

```
autoconf-2.59
automake-1.7.9
m4-1.4 (GNU m4)
```

• for automatically builted files:

```
ginac-1.1.6
flex-2.5.4
bison-1.875a
```

• for maintaining documentation:

```
makeinfo-4.6
texi2html-1.65
tex-3.14159
texinfo-4.6
latex 2e (2001/06/01)
dvips-5.92b
gnuplot-3.7
transfig-3.2
doxygen-1.2.13.1
```

• Version numbers are indicative: others versionnumbers sould also work. Nevertheless, these version numbers are known to work well.

• For participating to the rheolef concurrent development, please contact also `Pierre.Saramito@imag.fr` for concurrent development.

3 Reporting Bugs

This software is still under development. Please, run the `make check` non-regression tests. Send comments and bug repports 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.

4 Commands

4.1 `cad` - plot a boundary file

(Source file: '`nfem/bin/cad.cc`')

Description

Manipulates geometry boundary files (CAD, Computer Aid Design): visualizations and file conversions. This command is under development.

Synopsis

```
col < file[.qmgcad] | cad -input-qmg - options...
```

Examples

The `qmg` logo:

```
col < ../data/logo2.qmgcad  | cad -input-qmg - -noclean
gnuplot output.plot
```

A torus:

```
col < ../data/test2-out.qmgcad | cad -input-qmg -
```

In the `geomview` window, enter '2as' key sequence to switch to a smooth rendering.

Unresolved issues

The `qmg` demonstration files comes from PC-DOS and contains line-feed characters that are not filtered by `flex` when reading file. Thus, the `col` filter is required.

Included surfaces, such as cracks, are hiden in 3D with `geomview`. Thus, `vtk` rendering with some transparency factor would be better.

`bamg` and `grummp` boundary files are not yet supported. They will be in the future.

Input file specification

filename specifies the name of the file containing the input mesh. The ".cad" extension is assumed.

`-I`*caddir* add *caddir* to the mesh 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 (see Section 5.1 [cad class], page 41).

`-` read mesh on standard input instead on a file.

Input format options

`-input-cad`
: read the boundary in the '`.cad`' format. This is the default (TODO: not yet, uses qmg instead).

`-input-qmg`
: read the boundary in the '`.qmgcad`' format. In `qmg`, this file format is known as *brep*, that stands for boundary representation.

Output format options

`-cad` output boundary on standard output stream in cad text file format, instead of plotting it.

`-geomview`
: use geomview rendering tool. This is the default for tridimensional data.

`-gnuplot` use gnuplot rendering tool. This is the default for bidimensional data.

Render options

`-subdivide` *nsub*
: Subdivide each Bezier patch in *degree*nsub* linear sub-element. For rendering tools that does not support Bezier patches, such as `gnuplot` or `geomview`. Default is *nsub*=3 for tridimensional data and *nsub*=50 for bidimensional data.

`-no-bezier-adapt`
: Subdivide each Bezier patch in *nsub* instead of *degree*nsub*.

`-bezier-adapt`
: Subdivide each Bezier patch in of *degree*nsub*. This is the default.

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 conjuction with the "-noclean" command.

4.2 `geo` - plot a finite element mesh

(Source file: '`nfem/bin/xgeo.cc`')

Synopsis

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

Description

Plot a 2d or 3d finite element mesh.

The input file format is described with the `geo` class manual (see Section 5.2 [geo class], page 42).

Example

Enter commands as:

```
geo -vtk bielle.geo
geo -vtk -fill bielle.geo
```

or, for 2D meshes:

```
geo -plotmtv square.geo
```

Input file specification

filename specifies the name of the file containing the input mesh. The ".geo" extension is assumed.

`-I`*geodir* add *geodir* to the mesh 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 (see Section 5.2 [geo class], page 42).

`-` 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 to change this default. Useful when dealing with output formats (graphic, format conversion) that creates auxilliary files, based on this name.

Inquire options

`-size` print the number of elements in the mesh and exit.

`-n-node` print the number of nodes in the mesh and exit.

`-hmin` print the smallest edge length in the mesh and exit.

`-hmax` print the largest edge length in the mesh and exit.

`-xmin` print the lower-left-bottom vertice in the mesh and exit.

`-xmax` print the upper-right-top vertice in the mesh and exit.

Render options

The default render could also specified by using the RHEOLEF_RENDER environment variable.

`-mayavi` use Mayavi. This is the default.
`-plotmtv` use plotmtv tool.
`-gnuplot` use gnuplot tool.
`-vtk` use Visualization Toolkit.
`-x3d` use x3d visualization tool.
`-atom` use PlotM atom representation (from lassp, Cornell).

Graphic options

`-color` use colors. This is the default.
`-black-and-white`
Option only available with `mayavi`.
`-stereo` Rendering mode suitable for red-blue anaglyph 3D stereoscopic glasses. Option only available with `mayavi`.
`-grid` rendering mode. This is the default.
`-fill` rendering mode. Fill mesh faces using light effects when available.
`-shrink` rendering mode. Shrink 3d elements (with vtk only).
`-tube` rendering mode. All edges appears as tubes (with vtk only).
`-ball` rendering mode. Vertices appears as balls (with vtk only).
`-full` rendering mode. All edges appears (with vtk only).
`-cut` rendering mode. cut by plane and clip (with vtk only). Works with '`-full`', '`-tube`' or '`-ball`' modes. Does not clip mesh with '`-fill`' and '`-grid`' modes. Does not work yet with '`-shrink`' mode. This option is still in development.
`-origin` *x* [*y* [*z*]]
set the origin point of the cutting plane when using with the '`-cut`' option. Default is (0.5,0.5,0.5).
`-normal` *nx* [*ny* [*nz*]]
set the normal vector of the cutting plane. when using with the '`-cut`' option. Default is (1,0,0).
`-split` split each edge by inserting a node. The resulting mesh is P2-iso-P1.

Output format options

`-geo` output mesh on standard output stream in geo text file format, instead of plotting it.
`-bamg` output mesh on standard output stream in bamg text file format, instead of plotting it.
`-gmsh` output mesh on standard output stream in gmsh ascii file format, instead of plotting it.
`-tegen` output mesh on stream in tegen text file format, instead of plotting it (TODO).
`-mmg3d` output mesh on standard output stream in mmg3d text file format, instead of plotting it.

`-cemagref`
: output mesh on standard output stream in cemagref text file format, instead of plotting it. Notices that the mesh does not contains the topographic elevation. If you want to include the elevation, in the output, use the `field` command with the `-cemagref` option.

`-upgrade` edges and faces are numbered. This numbering is required for the quadratic approximation.

`-ndigit` *int*
: number of digits used to print floating point values when using the '`-geo`' option. Default depends upon the machine precision of the `Float` type.

`-output-as-double`
: Avoid variation of output float formats when using high precision `doubledouble` or `bigfloat` classes. This option is used for non-regression test purpose.

Input format options

`-input-geo`
: read the mesh in the '`.geo`' format. This is the default.

`-input-bamg`
: read the mesh in the '`.bamg`' format. Since edges are not numbered in '`.bamg`' format, this file format is not suitable for using P2 elements. This format is supported only for format conversion purpose. See `-upgrade` for edge numbering while converting to '`.geo`'.

`-input-gmsh`
: read the mesh in the '`.msh`' format. Since edges in 2D and faces in 3D are not numbered in '`.msh`' format, this file format is not suitable for using P2 elements. This format is supported only for format conversion purpose. See `-upgrade` for edge numbering while converting to '`.geo`'.

`-input-tetgen`
: read the mesh as the concatenation of '`.node`', '`.ele`' and '`.face`' tetgen format. Since in 3D are not all numbered in tetgen format, this file format is not suitable for using P2 elements. This format is supported only for format conversion purpose. See `-upgrade` for edge numbering while converting to '`.geo`'.

`-input-mmg3d`
: read the mesh in the '`.mmg3d`' format. Since edges and faces are not numbered in '`.mmg3d`' format, this file format is not suitable for using P2 elements. This format is supported only for format conversion purpose. See `-upgrade` for edge numbering while converting to '`.geo`'.

`-input-grummp`
: read the mesh on standard input stream in grummp text file format. The `.g` grummp file format is defined grummp '`.template`' specification file

```
newfile g
"grummp"
"2" nverts ncells nbfaces
verts: coords
```

```
cells: verts region
bdryfaces: verts bc
```

Conversely, for tridimensionnal meshes, the grummp '`.template`' specification file is:

```
newfile g
"grummp"
"3" nverts ncells nbfaces
verts: coords
cells: verts region
bdryfaces: verts bc
```

Such files can be generated with the `tri` and `tetra` grummp mesh file generators. The grummp mesh generators suport multi-regions geometries. These regions are translated to bi- or tri-dimensional domains, that are no boundary domains. A file format conversion writes:

```
geo -input-grummp -geo -upgrade -noverbose - < hex-multi.g > h
```

`-input-qmg`

output mesh on standard output stream in qmg text file format. instead of plotting it. Since edges are not numbered in '`.qmg`' format, this file format is not suitable for using P2 elements. This format is supported only for format conversion purpose. See `-upgrade` for edge numbering while converting to '`.geo`'.

`-dmn` *domain-names-file*

An auxilliary '`.dmn`' file defining the boundary domain names as used by `rheolef` could be used, since `bamg`, `mmg3d`, `tetgen` and `grummp` use numeric labels for domains. Example of a '`multi-domain.dmn`' with two regions and four boundary domains file:

```
EdgeDomainNames
  4
  bottom
  right
  top
  left
FaceDomainNames
  2
  hard
  soft
```

`geo` recognized this format as an extension at the end of the '`.bamg`', `.mesh` (mmg3d) or the '`.g`' file. The complete file format conversion writes:

```
cat multi-domain.g multi-domain.dmn | \\
  geo -input-grummp -upgrade -geo -  > multi-domain.geo
```

If this file is not provided, boundary domains are named `boundary1`, `boundary2`, ... and region domain are named `region1`, `region2`, ... The tridimensionnal domain name file contains the `FaceDomainNames` and `VolumeDomainNames` keywords, for boundary and region domains, respectively.

`-input-cemagref`
: read the mesh in the '`.cemagref`' format. Notices that the reader skip the topographic elevation informations. If you want to load the elevation, use the `field` command with the `-input-cemagref` option.

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 conjuction with the "-noclean" command.

`-check`

`-dump` used for debug purpose.

Graphic output file

You can generate a `postscript`, `latex` or `Fig` output of your mesh, using the `geo` command with '`-gnuplot -noclean`' options, and then modifying the '`.plot`' generated file.

Geo file format

This is the default mesh file format. It contains two entitiess, namely a mesh and a list of domains. The mesh entity starts with the header. The header contains the `mesh` keyword followed by a line containing a format version number, the space dimension (e.g. 1, 2 or 3), the number of vertices, the number of elements. For file format version 2, when dimension is three, the number of faces is specified, and then, when dimension is two or three, the number of edges is also specified. Follows the vertex coordinates, the elements and the faces or edges. One element starts with a letter indicating the element type

'`p`' point

'`e`' edge

'`t`' triangle

'`q`' quadrangle

'`T`' tetrahedron

'`P`' prism

'`H`' hexaedron

Then, we have the vertex indexes. A vertex index is numbered in the C-style, i.e. the first index is numbered 0. An edge is a couple of index.

Geo version 1 file format (obsolete) does neither give any information about edges for 2d meshes nor for faces in 3d. This information is requiered for maintaining P2 approximation file in a consistent way, since edge and face numbering algorithm should vary from one version to another, and thus may be stored on file.

Nevertheless, geo version 1 file format is still supported, since it is a convenient way to build and import meshes (see Section 4.2 [geo command], page 16).

A sample mesh is in version 1 writes

```
mesh
1 2 4 2
0 0
1 0
1 1
0 1
t   0 1 3
t   1 2 3
```

the same in version 2

```
mesh
2 2 4 2 5
0 0
1 0
1 1
0 1
t   0 1 3
t   1 2 3
0 1
1 2
2 3
3 0
1 3
```

The second entity is a list of domains, that finishes with the end of file. A domain starts with the `domain` keyword, followed by a domain name. Then, the format version number (i.e. 1 today), and the number of sides. Follows the sides

```
domain
bottom
1 1 1
e   0 1

domain
top
1 1 1
e   2 3
```

4.3 `space` - print a finite element space

(Source file: '`nfem/bin/xspace.cc`')

Synopsis

`space` *filename*`[.space]`

Description

Read and re-output a finite element space.

Example

enter command as:

```
space < my.space
```

Space format

This is the default space file format. The file starts with the header: the "space" keyword followed by a line containing a format version number (i.e. 1 today), and the number of degrees of freedom. Follows the numbering and an optional 'B' label, if the degree of freedom is blocked. A sample space is:

```
space
1 3
0 B
1 B
0
2 B
1
```

4.4 `field` – plot a field

(Source file: '`nfem/bin/xfield.cc`')

Synopsis

`field [-I`*dir*`]` *filename*

Description

Read and output a finite element field from file.

Example

```
field square.field
field square.field -black-and-white
field box.field
field box.field -volume
```

Input file specification

`-I`*dir* add *dir* to the RHEOPATH search path. See also Section 5.2 [geo class], page 42 for RHEOPATH mechanism.

filename specifies the name of the file containing the input field.

`-` read field on standard input instead on a file.

`-name` when field data comes from standard input, the field name is not known and is set to `"output"` by default. This option allows to change this default. Useful when dealing with output formats (graphic, format conversion) that creates auxilliary files, based on this name.

Input format options

`-input-field`
: read the data in the '`.field`' format. This is the default.

`-input-cemagref`
: input data in cemagref file format. The field represents the topographic elevation and is assumed to be a `P1` approximation.

Output format options

`-field`

`-text` output field on standard output stream in field text file format.

`-bamg` output field on standard output stream in bamg file format.

`-mmg3d` output field on standard output stream in mmg3d file format.

`-gmsh` output field on standard output stream in gmsh file format.

`-cemagref`
: output field on standard output stream in cemagref file format. The field represents the topographic elevation and is assumed to be a `P1` approximation. This option is suitable for file format conversion:

`-ndigit` *int*
: number of digits used to print floating point values. Default depends upon the machine precision associated to the `Float` type.

`-round` Round the output, in order for the non-regression tests to be portable across platforms. Floating point are machine-dependent.

File format conversions

Input and output options can be combined for efficient file format conversion:

```
field -input-cemagref -field - < mytopo.cemagref > mytopo.field
field myfile.field -bamg > myfile.bb
```

can be performed in one pass:

```
field -input-cemagref -bamg  - < mytopo.cemagref > myfile.bb
```

Getting information

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

Render options

The default render could also specified by using the RHEOLEF_RENDER environment variable.

`-gnuplot` use `gnuplot` tool. This is the default for monodimensional geometries.
`-mayavi` use `mayavi` tool. This is the default for bi- and tridimensional geometries.
`-plotmtv` use `plotmtv` plotting command, for bidimensional geometries.
`-vtk` use `vtk` tool. This is the old (no more supported) tool for tridimensional geometries. Could be used in 2d also.

Rendering options

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

`-stereo`
`-nostereo`
rendering mode: suitable for red-blue anaglyph 3D stereoscopic glasses.

`-fill` isoline intervals are filled with color. This is the default.
`-nofill` draw isolines by using lines.
`-grid` mesh edges appear also.
`-nogrid` prevent mesh edges drawing. This is the default.
`-proj` Convert all selected fields to P1-continuous approximation by using a L2 projection. Usefull when input is a discontinuous field (e.g. P0 or P1d).
`-iso` *float* extract an iso-value set of points (1d), lines (2d) or surfaces (3d) for a specified value. Output the surface in text format (with `-text` option) or using a graphic driver. This option requires the `vtk` code.
`-noiso` do not draw isosurface.
`-volume`
`-novolume`
This is an experimental volume representation by using ray cast (mayavi and vtk graphics).

`-n-iso` *int*
For 2D visualizations, the isovalue table contains regularly spaced values from fmin to fmax, the bounds of the field.

`-n-iso-negative` *int*
The isovalue table is splitted 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 usefull when plotting e.g. vorticity or stream functions, where the sign of the field is representative.

`-label`
`-nolabel` Write or do not write labels for values on isolines on 2d contour plots when using `plotmtv` output.

`elevation`
For two dimensional field, represent values as elevation in the third dimension.

`noelevation`
Prevent from the `elevation` representation. This is the default.

`-scale` *float*
applies a multiplicative factor to the field. This is useful e.g. in conjonction with the `elevation` option. The default value is 1.

`-cut` show a cut by a specified plane. The cutting plane is specified by its origin point and normal vector. This option requires the `vtk` 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).

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 conjuction 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 finaly the list of values: A sample field file (compatible with the sample mesh example presented in command manual; see Section 4.2 [geo command], page 16) 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 -plotmtv
```

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.

4.5 `mfield` – handle multi-field files

(Source file: '`nfem/bin/mfield.cc`')

Synopsis

`mfield {-`*field-name*`}+ [-I`*dir*`]` *filename*

Example

This command performs a *velocity* graphical output for vector-valued fields:

```
mfield -u square.mfield -velocity
mfield -u square.mfield -deformation
mfield -s square.mfield -proj -tensor
```

It is also convenient for selecting some scalar fields, e.g. to pipe to another processor

```
mfield -psi square.mfield | field -
mfield -u0  square.mfield | field -
mfield -u1  square.mfield | field -
```

Description

Read and output multiple finite element fields from file, in field text file format. Fields are preceded by a label, in the following '`.mfield`' file format:

```
mfield
1 2
#u0
  field u0...
#u1
  field u1...
#p
  field p...
#psi
  field psi...
```

Such labeled file format could be easily generated using the `catchmark` stream manipulator (see Section 5.21 [iorheo class], page 79).

```
cout << catchmark("u")   << uh
     << catchmark("p")   << ph
     << catchmark("psi") << psih;
```

Options

`-I`*dir* add *dir* to the RHEOPATH search path. See also Section 5.2 [geo class], page 42 for RHEOPATH mechanism.

filename specifies the name of the file containing the input field.

`-all` all fields are selected for stdout printing.

`-` 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.

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

`-noverbose`
does not print previous messages.

`-round` [*float*]
Round the output, in order for the non-regression tests to be portable across platforms. Floating point are machine-dependent. The round value is optional. default value is 1e-10.

Render options

`-velocity`
: Render vector-valued fields as arrows using `plotmtv`, `mayavi` or `vtk`. The the numeric extension is assumed: use e.g. '`-u`' option instead of '`-u0`', '`-u1`' options.

`-deformation`
: Render vector-valued fields as deformed mesh using `plotmtv`, `mayavi` or `vtk`. The the numeric extension is assumed: use e.g. '`-u`' option instead of '`-u0`', '`-u1`' options.

`-tensor`
: Render tensor-valued fields as ellipses using `mayavi`. The the numeric extension is assumed: use e.g. '`-tau`' option instead of '`-tau00`', '`-tau01`'... Warning: in development stage.

`-vscale` *float*
: scale vector values by specified factor. Default value is 1.

`-proj`
: Convert all selected fields to P1-continuous approximation by using a L2 projection. Only valid yet together with the `-tensor` option.

Render tool selection

`-mayavi`
: use `mayavi` tool. This is the default for bi- and tridimensional geometries. The environment variable RHEOLEF_RENDER may be set to one of the choices below to override this default.

`-plotmtv`
: use `plotmtv` plotting command.

`-vtk`
: use `vtk` tool. Old style for 3d problems: superseted by `-mayavi`.

Others render options

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

`-stereo`
`-nostereo`
: rendering mode: suitable for red-blue anaglyph 3D stereoscopic glasses.

`-fill`
: vector norm isolines intervals are filled with color.

`-nofill`
: vector norm isolines are drawed by using colored mesh edges. This is the default.

`-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 conjuction with the `-noclean` command.

To do

Check if selected fields appear in file. Otherwise, send some error message.

4.6 `branch` – handle a family of fields

(Source file: '`nfem/bin/branch.cc`')

Synopsis

`branch` [*options*] *filename*

Example

Generates vtk file colection 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

`-I`*dir* add *dir* to the RHEOPATH search path. See also Section 5.2 [geo class], page 42 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.

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

`-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 requieres `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 conjonction 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 usefull in conjuction 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
  ...
```

4.7 bamg2geo - convert bamg mesh in geo format

(Source file: '`nfem/bin/bamg2geo`')

Synopsis

`bamg2geo` *options* *input*`[.bamg]` *input*`[.dmn]`
`bamg2geo` *options* *input*`[.bamg] -Cl` *domlabel*
`bamg2geo` *options* *input*`[.bamg] {-dom` *domname*`}*`

Description

Convert a bamg '`.bamg`' into '`.geo`' one. The output goes to standart output. The '`.dmn`' file specifies the domain names, since `bamg` mesh generator uses numbers as domain labels.

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 auxilliary '`.dmn`' file defines the boundary domain names as used by Rheolef, since `bamg` 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 usefull for some special boundary conditions.

Options

`-upgrade`
`-noupgrade`
: Default is to output a version 2 '`.geo`' file format. See Section 4.2 [geo command], page 16. With the `-noupgrade`, a version 1 file format is assumed.

`-dom` *dom1* `...` `-dom` *domN*
`-Cl {poi|sed|sec|tro|con|NONE}`
: Predefined domain name convention. See next section.

4.8 mesh2geo - convert mmg3d mesh in geo format

(Source file: '`nfem/bin/mesh2geo`')

Synopsis

`mesh2geo` *options* *input*`[.mesh]` *input*`[.dmn]`

Description

Convert a mesh '`.mesh`' into '`.geo`' one. The output goes to standart output. The '`.dmn`' file specifies the domain names, since `mmg3d` mesh generator uses numbers as domain labels.

Example

```
mmg3d -O 1 -in toto.mesh -out toto-opt.mesh
mesh2geo toto-opt.mesh toto.dmn > toto-opt.geo
```

Domain name file

This auxilliary '`.dmn`' file defines the boundary domain names as used by Rheolef, since `mmg3d` uses numeric labels for domains.

Options

`-upgrade`
`-noupgrade`
: Default is to output a version 2 '`.geo`' file format. See Section 4.2 [geo command], page 16. With the `-noupgrade`, a version 1 file format is assumed.

4.9 tetgen2geo - convert tetgen mesh in geo format

(Source file: '`nfem/bin/tetgen2geo`')

Synopsis

`tetgen2geo` *options* *input*`[.node]` *input*`[.ele]` *input*`[.face]` *input*`[.dmn]`

Description

Convert a tetgen mesh into '`.geo`' one. The output goes to standart output. The '`.dmn`' file specifies the domain names, since `tetgen` mesh generator uses numbers as domain labels.

Example

```
tetgen toto.smesh
tetgen2geo toto.1.node toto.1.ele toto.1.face toto.dmn > toto.geo
```

Domain name file

This auxilliary '`.dmn`' file defines the boundary domain names as used by Rheolef, since `tetgen` uses numeric labels for domains.

Options

`-upgrade`
`-noupgrade`
: Default is to output a version 2 '`.geo`' file format. See Section 4.2 [geo command], page 16. With the `-noupgrade`, a version 1 file format is assumed.

4.10 msh2geo - convert gmsh mesh in geo format

(Source file: '`nfem/bin/msh2geo`')

Synopsis

`msh2geo` *options* *input*`[.msh]`

Description

Convert a gmsh '`.msh`' into '`.geo`' one. The output goes to standart output.

Example

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

See the `gmsh` documentation for a detailed description of the '`.mshcad`' input file for `gmsh`.

4.11 grummp2geo - convert grummp mesh in geo format

(Source file: '`nfem/bin/grummp2geo`')

Synopsis

`grummp` *options* *input*`[.g]` [*input*`[.dmn]`]

Description

Convert a grummp '`.g`' into '`.geo`' one. The output goes to standart output. The '`.dmn`' file specifies the domain names, since `grummp` mesh generators use numbers as domain labels.

Example

```
grummp2geo toto.g toto.dmn > toto.geo
```

4.12 qmg2geo - convert qmg mesh in geo format

(Source file: '`nfem/bin/qmg2geo`')

Description

Convert a qmg '`.qmg`' into '`.geo`' one. The output goes to standart output.

Synopsis

`qmg2geo [-qmg]` *input*`.qmg` *input*`.dmn`

Domain name file

This auxilliary '`.dmn`' file defines the boundary domain names as used by Rheolef, since `qmg` uses numeric labels for domains.

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

4.13 cemagref2field - convert cemagref topography in field and geo formats

(Source file: '`nfem/bin/cemagref2field`')

Synopsis

`cemagref2field` *options* *input*`[.cemagref]`

Description

Convert a cemagref '`.cemagref`' topography data file into '`.field`' and its associated '`.geo`' files. The '`.field`' output goes to standart output, while the compressed '`.geo`' is created as *input*`.geo.gz` using `gzip`.

Example

```
cemagref2field toto.cemagref > toto-z.field
cemagref2field -cemagref toto.dat > toto-z.field
```

and a '`toto.geo.gz`'.

Options

`-cemagref` *input*
: specifies directly the input file, for an alternate suffix convention.

4.14 mkgeo_grid – build a regular grid mesh in 1d, 2d or 3d

(Source file: '`nfem/bin/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 comand line:

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

Description

This command is usefull 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]x[c,d] rectangle or [a,b]x[c,d]x[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]x[-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

`-boundary`
: The boundary domains for boxes uses names: `left`, `right`, `top`, `bottom`,`front` and `back`. Instead of splitting the boundary in faces, this option groups all boundary faces in one domain named `boundary`.

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

Regions

`-region` The whole domain is splitted into two subdomains: `east` and `west`, 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` The corners (four in 2D and eight in 3D) are defined as OD-domains. This could be usefull 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 independant. The coordinate system is specified in the geometry file '`.geo`'.

`-rz` the 2d mesh is axisymmetric.

4.15 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 usefull 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 Chapter 2 [Installing], page 3. The `rheolef-config` command returns these settings.

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

Another usefull 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, especialy 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.

5 Classes

5.1 `cad` - the boundary definition class

(Source file: '`nfem/lib/cad.h`')

Description

The `cad` class defines a container for the description of the boundary of a geometry, by using Bezier patches.

The aim is to handle high-level polynomial interpolation together with curved boundaries (non-polynomial). So, the description bases on Bezier curves and surfaces.

Also, the adaptive mesh loop requieres interpolation on the boundary, and this class should facilitates such procedures.

This class is actually under development.

File format

Under definition.

File format conversion

Under development. Conversion to geomview and vtk for visualization. Also to and from qmg, grummp, modulef/ghs, opencascade for compatibility.

Implementation

```
class cad : public smart_pointer<cad_rep> {
public:
// typedefs:

        typedef cad_rep::size_type size_type;


// allocator:

        cad();
        cad (const std::string& file_name);

// accessors:

        size_type dimension() const;
        size_type n_face() const;

        const point& xmin() const;
        const point& xmax() const;
```

```
        void  eval (const cad_element& S, const point& ref_coord, point & real_coor
        point eval (const cad_element& S, const point& ref_coord) const;

// input/output:

        friend std::istream& operator >> (std::istream& is, cad& x);
        friend std::ostream& operator << (std::ostream& os, const cad& x);
};
```

5.2 geo - the finite element mesh class

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

Synopsys

The `geo` class defines a container for a finite element mesh. This describes the nodes coordinates and the connectivity. `geo` can contains domains, usefull for boundary condition setting.

Example

A sample usage of the geo class writes

```
geo omega;
cin >> omega;
cout << mayavi << full << omega;
```

Description

The empty constructor makes an empty geo. A string argument, as

```
geo g("square");
```

will recursively look for a '`square.geo[.gz]`' file in the directory mentionned by the RHEOPATH environment variable, while `gzip` decompression is assumed. If the file starts with '`.`' as '`./square`' or with a '`/`' as in '`/home/oscar/square`', no search occurs. Also, if the environment variable RHEOPATH is not set, the default value is the current directory.

Input and output on streams are available, and manipulators works for text or graphic output (see Section 4.2 [geo command], page 16).

Finally, an STL-like interface provides efficient accesses to nodes, elements and domains.

Mesh adaptation

The `geo_adapt` functions performs a mesh adaptation to improve the `P1`-Lagrange interpolation of the `field` given in argument (see Section 5.5 [field class], page 54).

Axisymetric geometry

The `coordinate_system` and `set_coordinate_system` members supports both `cartesian`, `rz` and `zr` (axisymmetric) coordinate systems. This information is used by the `form` class (see Section 5.7 [form class], page 56).

Access to connectivity

The folowing code prints triangle vertex numbers

```
geo omega ("circle");
for (geo::const_iterator i = g.begin(); i != i.end(); i++) {
  const geo_element& K = *i;
  if (K.name() != 't') continue;
  for (geo::size_type j = 0; j < 3; j++)
    cout << K [j] << " ";
  cout << endl;
}
```

See Section 8.9 [geo_element internal], page 125.

Access to vertice coordinates

The folowing code prints vertices coordinate

```
for (geo::const_iterator_vertex i = g.begin_vertex(); i != g.end_node(); i++) {
  const point& xi = *i;
  for (geo::size_type j = 0; j < g.dimension(); j++)
    cout << xi [j] << " ";
  cout << endl;
}
```

Access to domains

The following code prints edges on domain:

```
for (geo::const_iterator_domain i = g.begin_domain(); i != i.end_domain(); i++) {
  const domain& dom = *i;
  if (dom.dimension() != 2) continue;
  for (domain::const_iterator j = dom.begin(); j < dom.end(); j++) {
    const geo_element& E = *j;
    cout << E [0] << " " << E[1] << endl;
  }
  cout << endl;
}
```

See Section 5.3 [domain class], page 48.

Environs

RHEOPATH: search path for geo data file. Also Section 5.7 [form class], page 56.

Implementation

```
class geo : public smart_pointer<georep> {
public:

// typedefs:
```

```
  typedef georep::plot_options          plot_options;
  void write_gnuplot_postscript_options (std::ostream& plot, const plot_options& op

  typedef georep::iterator              iterator;
  typedef georep::const_iterator        const_iterator;
  typedef georep::elemlist_type         elemlist_type;
  typedef georep::nodelist_type         nodelist_type;
  typedef georep::size_type             size_type;
  typedef georep::domlist_type          domlist_type;

  typedef georep::const_iterator_node   const_iterator_node;
  typedef georep::const_iterator_vertex const_iterator_vertex;
  typedef georep::const_iterator_domain const_iterator_domain;
  typedef georep::iterator_node         iterator_node;
  typedef georep::iterator_vertex       iterator_vertex;
  typedef georep::iterator_domain       iterator_domain;

// allocators/deallocators:

  explicit geo (const std::string& filename, const std::string& coord_sys = "cartes
  geo(const geo& g, const domain& d);
  geo(const nodelist_type& p, const geo& g);
  geo();

  friend geo geo_adapt (const class field& criteria, const Float& hcoef,
        bool reinterpolate_criteria = false);

  friend geo geo_adapt (const class field& criteria,
        const adapt_option_type& = adapt_option_type(),
        bool reinterpolate_criteria = false);

  friend geo geo_metric_adapt (const field& mh,
        const adapt_option_type& = adapt_option_type());

// input/output:

  friend std::istream& operator >> (std::istream&, geo&);
  friend std::ostream& operator << (std::ostream&, const geo&);
  void save () const;
  void use_double_precision_in_saving();
  std::ostream& dump (std::ostream& s = std::cerr) const;
  std::ostream& put_mtv_domains (std::ostream&, size_type=0) const;

// accessors:

  const point&       vertex  (size_type i) const;
  const geo_element& element (size_type K_idx) const;
  Float              measure (const geo_element& K) const;
```

```
std::string name() const;
//  Family name plus number
std::string basename() const;
//  For moving boundary problems
std::string familyname() const;
//  Number of moving boundary mesh
size_type number() const;
//  Refinment iteration for the current mesh number
size_type version() const;
size_type dimension() const;
size_type map_dimension() const;
std::string coordinate_system () const; // "cartesian", "rz", "zr"
fem_helper::coordinate_type coordinate_system_type() const;

size_type serial_number() const;

const domain& boundary() const;
void build_subregion(const domain& start_from, const domain& dont_cross,
      std::string name, std::string name_of_complement="");
//  Builds a new domain on the side of domain 'dont_cross' on which 'start_from'
//  lies. These must have no intersection.

size_type size() const;
size_type n_element() const;  // same as size()
size_type n_vertex() const;
size_type n_vertice() const;
size_type n_node() const;     // same as n_vertex()
size_type n_edge() const ;
size_type n_face() const ;
size_type n_triangle() const ;
size_type n_quadrangle() const ;
size_type n_volume() const ;
size_type n_tetraedra() const ;
size_type n_prism() const ;
size_type n_hexaedra() const ;
size_type n_subgeo(size_type d) const ;
size_type n_element(reference_element::enum_type t) const;

Float hmin() const;
Float hmax() const;
const point& xmin() const;
const point& xmax() const;

meshpoint hatter (const point& x, size_type K) const;
point dehatter (const meshpoint& S) const;
point dehatter (const point& x_hat, size_type e) const;

const_iterator begin() const;
const_iterator end() const;
```

```
const_iterator_node begin_node() const;
const_iterator_node end_node() const;
const_iterator_vertex begin_vertex() const;
const_iterator_vertex end_vertex() const;

// localizer:
bool localize (const point& x, geo_element::size_type& element) const;
void localize_nearest (const point& x, point& y, geo_element::size_type& element)
bool trace (const point& x0, const point& v, point& x, Float& t, size_type& eleme

// access to domains:
const domain& get_domain(size_type i) const;
size_type n_domain() const;
bool has_domain (const std::string& domname) const;
const domain& operator[] (const std::string& domname) const;
const_iterator_domain begin_domain() const;
const_iterator_domain end_domain() const;

point normal (const geo_element& S) const;
point normal (const geo_element& K, georep::size_type side_idx) const;
void
sort_interface(const domain&, const interface&) const;
class field
normal (const class space& Nh, const domain& d,
      const std::string& region="") const;
class field
normal (const class space& Nh, const domain& d, const interface& bc) const;
class field
tangent (const class space& Nh, const domain& d,
      const std::string& region="") const;
class field
tangent (const class space& Nh, const domain& d, const interface& bc) const;
 //  Gives normal to d in discontinuous space Nh (P0 or P1d) and can
 //  initialize orientation of domain through 'bc' data structure.
 //  Currently limited to straight-sided elements.
class field
tangent_spline (const space& Nh, const domain& d, const interface& bc) const;
class field
plane_curvature (const space& Nh, const domain& d,
      const interface& bc) const;
 //  Gives curvature of d in the (x,y) or (r,z) plane.
 //  Currently limited to straight-sided elements.
class field
plane_curvature_spline (const space& Nh, const domain& d,
      const interface& bc) const;
 //  Gives curvature of d in the (x,y) or (r,z) plane based on a spline interpola
class field
plane_curvature_quadratic (const space& Nh, const domain& d,
      const interface& bc) const;
```

```
    //  Gives curvature of d in the (x,y) or (r,z) plane based on a local quadratic

  class field
  axisymmetric_curvature (const class space& Nh, const domain& d) const;
  class field
  axisymmetric_curvature (const class space& Nh, const domain& d,
        const interface& bc) const;
    //  Specific to "rz" and "zr" coordinate systems:
    //  Gives curvature of d in a plane orthogonal to the (r,z) plane.
    //  Can also initialize orientation of domain through 'bc' data structure.
    //  Currently limited to straight-sided elements.
  void
  interface_process (const domain& d, const interface& bc,
        geometric_event& event) const;
    //  Detects event along domain d sorted according to bc's.

  // construction of jump-interface domain data:
  void jump_interface(const domain& interface, const domain& subgeo,
        std::map<size_type, tiny_vector<size_type> >& special_elements,
        std::map<size_type,size_type>& node_global_to_interface) const;

// comparator:

  bool operator == (const geo&) const;
  bool operator != (const geo&) const;

// modifiers

  //  build from a list of vertex and elements:
  template <class ElementContainer, class VertexContainer>
  void build (const ElementContainer&, const VertexContainer&);

  void set_name (const std::string&);
  void set_coordinate_system (const std::string&); // "cartesian", "rz", "zr"
  void upgrade();
  void label_interface(const domain& dom1, const domain& dom2, const std::string& n

  // modify domains
  void erase_domain (const std::string& name);
  void insert_domain (const domain&);

// accessors to internals:
  bool localizer_initialized () const;
  void init_localizer (const domain& boundary, Float tolerance=-1, int list_size=0)
  const size_type* get_geo_counter() const;
  const size_type* get_element_counter() const;

  // TO BE REMOVED
  int gnuplot2d (const std::string& basename,
```

```
                plot_options& opt) const;

// check consistency

  void check()  const;

protected:

  friend class spacerep;
  friend class field;

  void may_have_version_2() const; // fatal if not !

// modifiers to internals:

  void set_dimension (size_type d);
  iterator begin();
  iterator end();
  iterator_node begin_node();
  iterator_node end_node();
  iterator_vertex begin_vertex();
  iterator_vertex end_vertex();
};
```

5.3 domain - part of a finite element mesh

(Source file: 'nfem/lib/domain.h')

Description

The domain class defines a container for a part of a finite element mesh. This describes the connectivity of edges or faces. This class is usefull for boundary condition setting.

Implementation

```
class domain : public Vector<geo_element> {
public:

// typedefs:

      typedef Vector<geo_element> sidelist_type;
      typedef Vector<geo_element>::size_type size_type;

// allocators/deallocators:

      domain(size_type sz = 0, const std::string& name = std::string());
      domain(const domain&);

// accessors:
```

```
      const std::string& name() const;
      size_type dimension() const;

// modifiers:

      void set_name(const std::string&);
      void set_dimension(size_type d);
      domain& operator=(const domain&);
      domain& operator += (const domain&);
      friend domain operator + (const domain&, const domain&);
      void resize(size_type n);
      void cat(const domain& d);
      template <class IteratorPair>
        void set(IteratorPair p, size_type n, const std::string& name);
      template <class IteratorElement>
        void set(IteratorElement p, size_type n, size_type dim, const std::string& n

//  input/ouput:

      friend std::ostream& operator << (std::ostream& s, const domain& d);
      friend std::istream& operator >> (std::istream& s, domain& d);
      std::ostream& put_vtk (std::ostream& vtk, Vector<point>::const_iterator first_
                Vector<point>::const_iterator last_p) const;
      std::ostream& dump (std::ostream& s = std::cerr) const;
      void check() const;

// data:
protected:
      std::string   _name;
      size_type     _dim;

};
```

5.4 space – piecewise polynomial finite element space

(Source file: ‘nfem/lib/space.h’)

Description

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

Degree of freedom numbering

Numbering of degrees of freedom (or shortly, *dof*) depends upon the mesh and the piecewise polynomial approximation used. This numbering is then used by the field and form classes. See also Section 5.5 [field class], page 54 and Section 5.7 [form class], page 56.

The degree-of-freedom (or shortly, *dof*) follows some simple rules, that depends of course of the approximation. These rules are suitable for easy `.field` file retrieval (see Section 5.5 [field class], page 54).

`P0`
`bubble` dof numbers follow element numbers in mesh.
`P1` dof numbers follow vertice numbers in mesh.
`P2` dof numbers related to vertices follow vertice numbers in mesh. Follow dof numbers related to edges, in the edge numbering order.
`P1d` dof numbers follow element numbers in mesh. In each element, we loop on vertices following the local vertice order provided in the mesh.

Unknown and blocked degree of freedom

A second numbering is related to the *unknown* and *blocked* degrees of freedom.

```
geo omega("square");
space V(omega,"P1");
V.block ("boundary");
```

Here, all degrees of freedom located on the domain `"boundary"` are marked as blocked.

File format

File output format for `space` is not of practical interest. This file format is provided for completness. Here is a small example

```
space
1 6
square
P1
   0 B
   1 B
   0
   1
   2 B
   2
```

where the '`B`' denotes blocked degres of freedom. The method `set_dof(K, dof_array)` gives the numbering in `dof_array`, supplying an element `K`. The method `index(dof)` gives the unknown or blocked numbering from the initial numbering, suppling a given `dof`

Additional features

The method `xdof(K, i)` gives the geometrical location of the i-th degree of freedom in the element `K`.

Product spaces, for non-scalar fields, such as vector-valued or tensor-valued (symmetric) fields,

Can also be defined

```
space U (omega, "P1", "vector");
space T (omega, "P1", "tensor");
```

Then, the number of component depends upon the geometrical dimension of the mesh `omega`.

Arbitrarily product spaces can also be constructed

```
space X = V0*V1;
```

Spaces for fields defined on a boundary domain can also be constructed

```
space W (omega, omega["top"], "P1");
```

The correspondance between the degree-of-freedom numbering for the space `W` and the global degree-of-freedom numbering for the space `V` is supported. The method `set_global_dof(S, dof_array)` gives the global numbering in `dof_array`, supplying a boundary element `S`. The method `domain_index(global_dof)` gives the unknown or blocked numbering from the initial numbering, suppling a given `global_dof`, related to the space `V`.

Implementation

```
class space : public smart_pointer<spacerep> {
public:
// typdefs:

    typedef spacerep::size_type size_type;

// allocator/deallocator:

    space ();
    space (const geo& g, const std::string& approx_name, const std::string& valued =
    space (const geo& g, const std::string& approx_name,
        const domain& interface, const domain& subgeo, const std::string& valued =
        //  For spaces with a discontinuity through domain 'interface', but otherwi
    space (const geo& g, const domain& d, const std::string& approx_name);
    space(const const_space_component&);

    space operator = (const const_space_component&);
    friend space operator * (const space&, const space&);
    friend space pow (const space&, size_type);

// modifiers:

    void block () const;
    void block_Lagrange () const;
    void block (size_type i_comp) const;
    void block_Lagrange (size_type i_comp) const;
    void block (const std::string& d_name) const;
    void block (const domain& d) const;
    void block (const std::string& d_name,  size_type i_comp) const;
    void block (const domain& d, size_type i_comp) const;
```

```
    void block_dof (size_type dof_idx, size_type i_comp) const;
    void block_dof (size_type dof_idx) const;
    void unblock_dof (size_type dof_idx, size_type i_comp) const;
    void unblock_dof (size_type dof_idx) const;
    void set_periodic (const domain& d1, const domain& d2) const;
    void set_periodic (const std::string& d1_name, const std::string& d2_name) cons
    void lock_components (const domain& d, point locked_dir) const;
    void lock_components (const std::string& d_name, point locked_dir) const;
    template <class Function>
    void lock_components (const std::string& d_name, Function locked_dir) const;
        //  Allows to enforce vector-boundary conditions.
        /*  Allows to enforce vector-boundary conditions.
         *!
         *! Current limitation: only 2D vector fields with components having same Fl
         *! The space has a dof for the direction normal to locked_dir, while the va
         *! direction locked_dir is blocked.
         *! The field::at function implements the reconstruction of the original ca
         */
    template <class Function>
    void lock_components (const domain& d, Function locked_dir) const;

// accessors:

    size_type  size()      const;
    size_type  degree ()   const;
    size_type  n_blocked() const;
    size_type  n_unknown() const;
    size_type  dimension() const;
    std::string coordinate_system() const;
    fem_helper::coordinate_type coordinate_system_type() const;


    const geo&                get_geo   () const;
    const basis&              get_basis (size_type i_component = 0) const;
    const numbering&          get_numbering (size_type i_component = 0) const;
    std::string               get_approx(size_type i_component = 0) const;
    const basis&              get_p1_transformation () const;

    std::string               get_valued() const;
    fem_helper::valued_field_type get_valued_type() const;

    size_type  n_component() const;
    space_component operator [] (size_type i_comp);
    const_space_component operator [] (size_type i_comp) const;

    size_type  size_component      (size_type i_component = 0) const;
    size_type  n_unknown_component (size_type i_component = 0) const;
    size_type  n_blocked_component (size_type i_component = 0) const;
    size_type  start               (size_type i_component = 0) const;
```

```
size_type  start_unknown       (size_type i_component = 0) const;
size_type  start_blocked       (size_type i_component = 0) const;

size_type  index              (size_type degree_of_freedom) const;
size_type  period_association (size_type degree_of_freedom) const;
bool       is_blocked         (size_type degree_of_freedom) const;
bool       is_periodic        (size_type degree_of_freedom) const;

bool       has_locked         () const;
             //  for (2D) vector spaces only, field is only blocked along locke
bool       is_locked          (size_type degree_of_freedom) const;
    //  for tangential-normal representation
size_type  locked_with  (size_type degree_of_freedom) const;
size_type  index_locked_with  (size_type degree_of_freedom) const;
    //  dof with the other component (thru space::index() as well)
Float      locked_component   (size_type dof, size_type i_comp) const;
    //  i-th component of locked direction
Float      unlocked_component (size_type dof, size_type i_comp) const;
    //  i-th component of unlocked direction

void freeze () const;
bool is_frozen() const;


// informations related to boundary spaces
bool  is_on_boundary_domain() const;
const domain& get_boundary_domain() const;
const geo&    get_global_geo     () const;
size_type  global_size () const;
size_type  domain_dof   (size_type global_dof) const;
size_type  domain_index (size_type global_dof) const;

// get indices of degrees of freedom associated to an element
    //  the tiny_vector is a "local index" -> "global index" table
    //      "global" variants differ only for boundary spaces (non-global uses
    //  mesh-wide indices)
void  set_dof        (const geo_element& K, tiny_vector<size_type>& idx) const;
void  set_global_dof (const geo_element& K, tiny_vector<size_type>& idx) const;
void  set_dof        (const geo_element& K, tiny_vector<size_type>& idx, size_t
void  set_global_dof (const geo_element& K, tiny_vector<size_type>& idx, size_t
void  set_dof        (const geo_element& K, tiny_vector<size_type>& idx, size_t
point x_dof      (const geo_element& K, size_type i_local) const;
 //  Hermite/Argyris: returns 1 if the global dof contains the derivative along
void  dof_orientation(const geo_element& K, tiny_vector<int>& orientation) cons

// piola transformation
meshpoint hatter (const point& x, size_type K) const;
point dehatter (const meshpoint& S) const;
point dehatter (const point& x_hat, size_type e) const;
```

```
// comparator

    bool operator == (const space&) const;
    bool operator != (const space&) const;

// input/output

   friend std::ostream& operator << (std::ostream&, const space&);
   friend std::istream& operator >> (std::istream&, space&);
   std::ostream& dump(std::ostream& s = std::cerr) const;
   void check() const;

// inquires:

   static size_type inquire_size(const geo&, const numbering&);

// implementation:
protected:
   space(smart_pointer<spacerep>);
   space (const space&, const space&); // X*Y
   space (const space&, size_type i_comp); // X[i_comp]
   friend class field;
};
```

5.5 `field` - piecewise polynomial finite element field

(Source file: '`nfem/lib/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 Section 5.4 [space class], page 49.

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_h ("circle");
space Vh (omega_h, "P1");
Vh.block ("boundary");
field uh (Vh);
uh ["boundary"] = 0;
```

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

```
Float u (const point&);
uh = interpolate (Vh, u);
```

Example

Here is a complete example using the field class:

```
Float u (const point& x) { return x[0]*x[1]; }
main() {
  geo omega_h ("square");
  space Vh (omega_h, "P1");
  field uh (Vh);
  uh = interpolate (Vh, u);
  cout << plotmtv << u;
}
```

Features

Algebra, such as x+y, x*y, x/y, lambda*x, ...are supported

Transformations applies to all values of a field:

```
field vh = compose(fabs, uh);
field wh = compose(atan2, uh, vh);
```

The composition supports also general unary and binary class-functions.

Vector-valued and tensor-valued field support is yet partial only. This feature will be more documented in the future.

5.6 `branch` - (t,uh

(Source file: '`nfem/lib/branch.h`')

Description

Stores a `field` together with its associated parameter value: a `branch` variable represents a pair (t,uh(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.

At each step, meshes are reloaded and spaces are rebuilted, even when they are identical.

The `vtk` render does not support mesh change during the loop (e.g. when using adaptive mesh process).

Implementation

```
class branch : public std::vector<std::pair<std::string,field> > {
public :

// allocators:

        branch ();
        branch (const std::string& parameter_name, const std::string& u_name);
        branch (const std::string& parameter_name, const std::string& u_name, const
        ~branch ();

// accessors:

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

// modifiers:

        void set_parameter (const Float& value);

// input/output:

        // get/set current value
        friend std::istream& operator >> (std::istream&, branch&);
        friend std::ostream& operator << (std::ostream&, const branch&);

        __branch_header         header ();
        __const_branch_header   header () const;
        __const_branch_finalize finalize () const;

        __obranch operator() (const Float& t, const field& u);
        __obranch operator() (const Float& t, const field& u, const field& p);

        __iobranch operator() (Float& t, field& u);
        __iobranch operator() (Float& t, field& u, field& p);
```

5.7 `form` - representation of a finite element operator

(Source file: '`nfem/lib/form.h`')

Description

The form class groups four sparse matrix, associated to a bilinear form on finite element space. This class is represented by four sparse matrix, associated to unknown and blocked degrees of freedom of origin and destination spaces (see Section 5.4 [space class], page 49).

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 m("square");
space V(m,"P1");
form a(V,V,"grad_grad");
field x(V);
x = fct;
field y = a*x;
cout << plotmtv << y;
```

This block-matrix operation is equivalent to:

```
y.u = a.uu*x.u + a.ub*x.b;
y.b = a.bu*x.u + a.bb*x.b;
```

Implementation

```
class form {
public :
// typedefs:

    typedef csr<Float>::size_type size_type;

// allocator/deallocator:

    form ();
    form (const space& X, const space& Y);
        //  locked_boundaries means that special vector BCs are applied,
        //  see space::lock_components()
    form (const space& X, const space& Y, const std::string& op_name,
        bool locked_boundaries=false);
    form (const space& X, const space& Y, const std::string& op_name, const domain&
        //  Currently weighted forms in P2 spaces use a quadrature formula which is
        //  limited to double precision floats.
    form (const space& X, const space& Y, const std::string& op_name, const field& w
        bool use_coordinate_system_weight=true);
    form (const space& X, const space& Y, const std::string& op_name,
        const domain& d, const field& wh, bool use_coordinate_system_weight=true);
    form (const space& X, const std::string& op_name);
    form (const form_diag& X);

// accessors:

    const geo&   get_geo() const;
    const space& get_first_space() const;
    const space& get_second_space() const;
    size_type nrow() const;
```

```
    size_type ncol() const;
    bool for_locked_boundaries() const { return _for_locked_boundaries; }

// linear algebra:

    Float operator () (const class field&, const class field&) const;
    friend class field operator * (const form& a, const class field& x);
    field trans_mult (const field& x) const;
    friend form trans(const form&);
    friend form operator * (const Float& lambda, const form&);
    friend form operator + (const form&, const form&);
    friend form operator - (const form&);
    friend form operator - (const form&, const form&);
    friend form operator * (const form&, const form&);
    friend form operator * (const form&, const form_diag&);
    friend form operator * (const form_diag&, const form&);

// input/output:

    friend std::ostream& operator << (std::ostream& s, const form& a);
    friend class form_manip;

// data
protected:
    space      X_;
    space      Y_;
    bool       _for_locked_boundaries;
public :
    csr<Float> uu;
    csr<Float> ub;
    csr<Float> bu;
    csr<Float> bb;
};
form form_nul(const space& X, const space& Y) ;
```

5.8 `form_diag` - diagional form class

(Source file: '`nfem/lib/form_diag.h`')

Description

Implement a form using two diagonal matrix.

Example

Here is the computation of the mass matrix for the P1 finite element:

```
geo m("square");
space V(m,"P1");
form_diag m(V,"mass");
```

Note

Implementation

```
class form_diag {
public:
// typedefs:

    typedef basic_diag<Float>::size_type size_type;

// allocator/deallocator:

    form_diag ();
    form_diag (const space& X);
    form_diag (const space& X, const char* op_name);
    form_diag (const class field& dh);

// accessors:

    const space& get_space() const;
    const geo&   get_geo() const;
    size_type size() const;
    size_type nrow() const;
    size_type ncol() const;

// linear algebra (partial):

    friend form_diag operator * (const form_diag&, const form_diag&);
    friend class field operator * (const form_diag& a, const class field& x);
    friend form_diag operator / (const Float& lambda, const form_diag& m);
    //new
    friend form_diag operator * (const Float& lambda, const form_diag&);
    friend form_diag operator + (const form_diag&, const form_diag&);
    friend form_diag operator - (const form_diag&);
    friend form_diag operator - (const form_diag&, const form_diag&);
    //wen

// input/output:

    friend std::ostream& operator << (std::ostream& s, const form_diag& a);

// data
private :
    space       X_;
public :
    basic_diag<Float> uu;
    basic_diag<Float> bb;
};
```

5.9 `trace` - restriction to boundary values operator

(Source file: '`nfem/lib/trace.h`')

Description

The `trace` operator restricts a field to its values located to a boundary value domain. The `trace` class is a derived class of the `form` class (see Section 5.7 [form class], page 56). Thus, all operations, such as linear algebra, that applies to `form`, applies also to `trace`.

Example

The following example shows how to plot the trace of a field, from standard input:

```
int main() {
    geo omega ("square");
    space Vh (omega "P1");
    space Wh (omega, "P1", omega ["top"]);
    trace gamma (Vh, Wh);

    field uh;
    cin >> uh;
    field wh (Wh);
    wh = gamma*uh;
    cout << wh;
}
```

Implementation

```
class trace : public form
{
public :

//  allocator/deallocator:

  trace();
  trace(const space& V, const space& W);

// accessors:

  const space& get_space() const;
  const space& get_boundary_space() const;
  const geo&   get_geo() const;
  const geo&   get_boundary_geo() const;
};
```

5.10 `form_manip` - form concatenation on a product space

(Source file: '`nfem/lib/form_manip.h`')

Description

build a form on a product space

To do

a general implementation without switch and with a loop.

Implementation

```
class form_manip {
public:
// typedefs:

    typedef form::size_type size_type;

// constructors/destructor:

    form_manip();

// accessors:

    size_type nform() const;
    size_type nrowbloc() const;
    size_type ncolbloc() const;
    form& bloc(size_type i, size_type j);
    const form& bloc(size_type i, size_type j) const;

// manipulators:

    form_manip& operator << (const form&);
    form_manip& operator >> (form&);
    form_manip& operator << (form_manip& (*)(form_manip&));

    friend form_manip& verbose (form_manip&);
    friend form_manip& noverbose (form_manip&);

// implementation:

    friend class size;
    friend form_manip& operator << (form_manip&, const class size&);

protected:
    bool         _verbose;
    size_type    _nform;
    size_type    _nrowbloc;
    size_type    _ncolbloc;
    std::vector<form> _a;
};
```

5.11 `form_diag_manip` - concatenation on a product space

(Source file: '`nfem/lib/form_diag_manip.h`')

Description

build a form_diag on a product space.

Example

The following example builds a 3-blocks diagonal form:

```
geo g("toto.geo");
space V(g,"P1");
space T = V*V*V;
form_diag Iv(V) ;
form_diag I(T) ;
form_manip I_manip;
I_manip << size(3) << Iv << Iv << Iv;
I_manip >> I;
```

Implementation

```
class form_diag_manip {
public:

// typedefs:

    typedef form_diag::size_type size_type;

// constructors/destructors:

    form_diag_manip();

// accessors:

    size_type nbloc() const;
    size_type nform() const;
    form_diag& bloc(size_type i);
    const form_diag& bloc(size_type i) const;

// manipulators:

    form_diag_manip& operator << (const form_diag& a);
    form_diag_manip& operator >> (form_diag& a);
    friend form_diag_manip& verbose (form_diag_manip &fm);
    friend form_diag_manip& noverbose (form_diag_manip &fm);
    form_diag_manip& operator<< (form_diag_manip& (*pf)(form_diag_manip&));
```

```
// implementation:

    friend class sized;
    friend form_diag_manip& operator << (form_diag_manip &fm, const class sized& s)

protected:
    bool              _verbose;
    size_type         _nform;
    size_type         _nbloc;
    std::vector<form_diag> _a;
};
```

5.12 point - vertex of a mesh

(Source file: 'nfem/basis/basic_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 basic_point {
    public:

// typedefs:

        typedef size_t size_type;
        typedef T      float_type;

// allocators:

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

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

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

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

```
// 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]; }


// inputs/outputs:

        std::istream& get (std::istream& s, int d = 3)
        {
            switch (d) {
            case 1 : _x[1] = _x[2] = 0; return s >> _x[0];
            case 2 : _x[2] = 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;

// ccomparators: lexicographic order

        template<int d>
        friend bool lexicographically_less (
                const basic_point<T>& a, const basic_point<T>& b) {
            for (size_type i = 0; i < d; i++) {
              if (a[i] < b[i]) return true;
              if (a[i] > b[i]) return false;
            }
            return false; // equality
        }
// algebra:

        friend bool operator == (const basic_point<T>& u, const basic_point<T>& v)
                { return u[0] == v[0] && u[1] == v[1] && u[2] == v[2]; }

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

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

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

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

```
        friend basic_point<T> operator+ (const basic_point<T>& u, const basic_point<
                { return basic_point<T> (u[0]+v[0], u[1]+v[1], u[2]+v[2]); }

        friend basic_point<T> operator- (const basic_point<T>& u)
                { return basic_point<T> (-u[0], -u[1], -u[2]); }

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

        template <class T1>
        friend basic_point<T> operator* (const T1& a, const basic_point<T>& u)
                { return basic_point<T> (a*u[0], a*u[1], a*u[2]); }

        friend basic_point<T> operator* (const basic_point<T>& u, T a)
                { return basic_point<T> (a*u[0], a*u[1], a*u[2]); }

        friend basic_point<T> operator/ (const basic_point<T>& u, const T& a)
                { return basic_point<T> (u[0]/a, u[1]/a, u[2]/a); }

        friend basic_point<T> operator/ (const basic_point<T>& u, basic_point<T> v)
                { return basic_point<T> (u[0]/v[0], u[1]/v[1], u[2]/v[2]); }

        friend basic_point<T> vect (const basic_point<T>& v, const basic_point<T>& 
                { return basic_point<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]); }
// metric:

        // TODO: non-constant metric
        friend T dot (const basic_point<T>& u, const basic_point<T>& v)
                { return u[0]*v[0]+u[1]*v[1]+u[2]*v[2]; }

        friend T norm2 (const basic_point<T>& u)
                { return dot(u,u); }

        friend T norm (const basic_point<T>& u)
                { return ::sqrt(norm2(u)); }

        friend T dist2 (const basic_point<T>& x,  const basic_point<T>& y)
                { return norm2(x-y); }

        friend T dist (const basic_point<T>& x,  const basic_point<T>& y)
                { return norm(x-y); }

        friend T dist_infty (const basic_point<T>& x,  const basic_point<T>& y)
                { return max(_my_abs(x[0]-y[0]),
                             max(_my_abs(x[1]-y[1]), _my_abs(x[2]-y[2]))); }
```

```
// data:
        T _x[3];
// internal:
protected:
        static T _my_abs(const T& x) { return (x > T(0)) ? x : -x; }
};
template <class T>
T vect2d (const basic_point<T>& v, const basic_point<T>& w);

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

// robust(exact) floating point predicates: return value as (0, > 0, < 0)
// formally: orient2d(a,b,x) = vect2d(a-x,b-x)
template <class T>
T orient2d(const basic_point<T>& a, const basic_point<T>& b,
        const basic_point<T>& x = basic_point<T>());

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

5.13 `tensor` - a N*N tensor, N=1,2,3

(Source file: '`nfem/basis/tensor.h`')

Synopsys

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

Implementation

```
class tensor {
    public:

        typedef size_t size_type;
        typedef Float  element_type;

// allocators:

        tensor (const Float& init_val = 0);
        tensor (Float x[3][3]);
        tensor (const tensor& a);

// affectation:
```

```
        tensor& operator = (const tensor& a);
        tensor& operator = (const Float& val);

    // modifiers:

        void fill (const Float& init_val);
        void reset ();
        void set_row    (const point& r, size_t i, size_t d = 3);
        void set_column (const point& c, size_t j, size_t d = 3);

    // accessors:

        Float& operator()(size_type i, size_type j);
        Float  operator()(size_type i, size_type j) const;
        point  row(size_type i) const;
        point  col(size_type i) const;
        size_t nrow() const; // = 3, for template matrix compatibility
        size_t ncol() const;

    // inputs/outputs:

        friend std::istream& operator >> (std::istream& in, tensor& a);
        friend std::ostream& operator << (std::ostream& out, const tensor& a);
        std::ostream& put (std::ostream& s, size_type d = 3) const;

    // algebra:

        friend bool  operator == (const tensor&, const tensor&);
        friend tensor operator - (const tensor&);
        friend tensor operator + (const tensor&, const tensor&);
        friend tensor operator - (const tensor&, const tensor&);
        friend tensor operator * (Float, const tensor&);
        friend tensor operator * (const tensor& a, Float k);
        friend tensor operator / (const tensor& a, Float k);
        friend point  operator * (const tensor& a, const point& x);
        friend point  operator * (const point& yt, const tensor& a);
               point  trans_mult (const point& x) const;
        friend tensor trans      (const tensor& a, size_type d = 3);
        friend tensor operator * (const tensor& a, const tensor& b);
        friend tensor inv        (const tensor& a, size_type d = 3);
        friend tensor diag (const point& d);

    // metric and geometric transformations:

        friend Float dotdot (const tensor&, const tensor&);
        friend Float norm2 (const tensor& a) { return dotdot(a,a); }
        friend Float dist2 (const tensor& a, const tensor& b) { return norm2(a-b); }
        friend Float norm  (const tensor& a) { return ::sqrt(norm2(a)); }
        friend Float dist  (const tensor& a, const tensor& b) { return norm(a-b); }
```

```
        Float determinant (size_type d = 3) const;
        friend Float determinant (const tensor& A, size_t d = 3);
        friend bool invert_3x3 (const tensor& A, tensor& result);

// spectral:
        // eigenvalues & eigenvectors:
        // a = q*d*q^T
        // a may be symmetric
        // where q=(q1,q2,q3) are eigenvectors in rows (othonormal matrix)
        // and   d=(d1,d2,d3) are eigenvalues, sorted in decreasing order d1 >= d2 >
        // return d
        point eig (tensor& q, size_t dim = 3) const;
        point 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 (othonormal matri
        //       v=(v1,v2,v3) are right pseudo-eigenvectors in rows (othonormal mati
        // and   s=(s1,s2,s3) are eigenvalues, sorted in decreasing order s1 >= s2 >
        // return s
        point svd (tensor& u, tensor& v, size_t dim = 3) const;

// data:
        Float _x[3][3];
// internal:
        std::istream& get (std::istream&);
};
// t = a otimes b
tensor otimes (const point& a, const point& b, size_t na = 3);
// t += a otimes b
void cumul_otimes (tensor& t, const point& a, const point& b, size_t na = 3);
void cumul_otimes (tensor& t, const point& a, const point& b, size_t na, size_t nb)
```

5.14 `vec` - dense vector

(Source file: '`skit/lib/vec.h`')

Description

The class implements an array. A declaration whithout any parametrers correspond to an array with a null size:

```
vec<Float> x;
```

Here, a `Float` array is specified. Note that the `Float` depends upon the configuration (see Chapter 2 [Installing], page 3). The constructor can be invocated whith a size parameter:

```
vec<Float> x(n);
```

Notes that the constructor can be invocated with an initializer:

```
vec<Float> y = x;
```

or

```
Float lambda = 1.3;
vec<Float> y = lambda;
```

Assignments are

```
x = y;
```

or

```
x = lambda;
```

Linear combinations are `x+y`, `x-y`, `x*lambda`, `lambda*x`, `x/lambda`.
Others combinations are `x*y`, `x/y`, `lambda/x`.
The euclidian norm is `norm(x)` while `dot(x,y)` denotes the euclidian scalar product,
Input/output routines overload `"<<"` and `">>"` operators:

```
cin  >> x;
```

and

```
cout << x;
```

See Section 5.21 [iorheo class], page 79.

Note

Since the `vec<T>` class derives from the `Array<T>` class, the `vec` class present also a STL-like container interface.
Actually, the `vec<T>` implementation uses a STL `vector<T>` class.

Implementation

```
template <class T>
class vec : public Array<T>
{
public:

    typedef          T                   element_type;
    typedef typename Array<T>::size_type size_type;

    // cstor, assignment
        explicit vec (unsigned int n = 0, const T& init_value = std::numeric_limits
          : Array<T>(n, init_value) {}
        vec<T> operator = (T lambda);
```

```
    // accessors
        unsigned int size () const { return Array<T>::size(); }
        unsigned int n () const { return size(); }

#ifdef _RHEOLEF_HAVE_EXPRESSION_TEMPLATE

        template <class X>
        vec(const VExpr<X>& x) : Array<T>(x.size()) { assign (*this, x); }

        template <class X>
        vec<T>& operator = (const VExpr<X>& x)
                                { logmodif(*this); return assign (*this, x); }
#endif // _RHEOLEF_HAVE_EXPRESSION_TEMPLATE
};
// io routines
template <class T> std::istream& operator >> (std::istream&, vec<T>&);
template <class T> std::ostream& operator << (std::ostream&, const vec<T>&);
```

5.15 `csr` - compressed sparse row matrix format

(Source file: '`skit/lib/csr.h`')

Description

The class implements a matrix in compressed sparse row format. A declaration whithout any parametrers correspond to a matrix with null size:

```
csr<double> a;
```

Notes that the constructor can be invocated with an initializer:

```
csr<double> a = b;
```

Input and output, as usual (see Section 5.21 [iorheo class], page 79):

```
cin  >> a;
cout << a;
```

Default is the Harwell-Boeing format. Various others formated options are available: matlab and postscript plots.

Affectation from a scalar

```
a = 3.14;
```

The matrix/vector multiply:

```
a*x
```

and the transposed matrix/ vector multiply:

```
a.trans_mult(x);
```

The binary operators are:

```
a*b, a+b, a-b, lambda*a, a*lambda, a/lambda
```

The unary operators are sign inversion and transposition:

```
-a, trans(a);
```

The combination with a diagonal matrix is not yet completely available. The interface would be something like:

```
basic_diag<double> d;
a+d, d+a, a-d, d-a
a*d, d*a,
a/d             // aij/djj
left_div(a,d)   // aij/dii
```

When applied to the matrix directly: this feature is not yet completely available. The interface would be something like:

```
a += d;         // a := a+d
a -= d;         // a := a-d
a *= d;         // a := a*d
a.left_mult(d); // a := d*a
a /= d;         // aij := aij/djj
a.left_div(d);  // aij := aij/dii
```

The combination with a constant-identity matrix: this feature is not yet completely available. The interface would be something like:

```
double k;
a + k*EYE, k*EYE + a, a - k*EYE, k*EYE - a,

a += e;
a -= e;
```

Get the lower triangular part:

```
csr<double> l = tril(a);
```

Conversely, `triu` get the upper triangular part.

For optimizing the profile storage, I could be convenient to use a renumbering algorithm (see also Section 5.16 [ssk class], page 72 and Section 5.18 [permutation class], page 76).

```
permutation p = gibbs(a);
csr<double> b = perm(a, p, q); // B := P*A*trans(Q)
```

Horizontal matrix concatenation:

$$a = \begin{pmatrix} a_{11} & a_{12} \end{pmatrix}$$

```
a = hcat(a11,a12);
```

Vertical matrix concatenation:

$$a = \begin{pmatrix} a_{11} \\ a_{21} \end{pmatrix}$$

```
a = vcat(a11,a21);
```

Explicit conversion from an associative `asr e` matrix writes:

```
a = csr<double>(e);
```

from a dense `dns m` matrix writes:

```
a = csr<double>(m);
```

Note

The `csr` class is currently under reconstruction for the distributed memory support by using a MPI-based implementation.

5.16 `ssk` - symmetric skyline matrix format

(Source file: '`skit/lib/ssk.h`')

Description

The class implements a symmetric matrix Choleski factorization. Let a be a square invertible matrix in `csr` format (see Section 5.15 [csr class], page 70).

```
csr<Float> a;
```

We get the factorization by:

```
ssk<Float> m = ldlt(a);
```

Each call to the direct solver for a*x = b writes either:

```
vec<Float> x = m.solve(b);
```

or

```
m.solve(b,x);
```

Data structure

The storage is either skyline, multi-file or chevron. This alternative depends upon the configuration (see Chapter 2 [Installing], page 3). The chevron data structure is related to the multifrontal algorithm and is implemented by using the spooles library. The multi-file data structure refers to the out-of-core algorithm and is implemented by using the taucs

library. If such a library is not available, the `ssk` class implements a skyline storage scheme and the profil storage of the `ssk` matrix is optimized by optimizing the renumbering, by using the Gibbs, Pooles and Stockmeyer algorithm available via the `permutation` class (see Section 5.18 [permutation class], page 76).

> Algorithm 582, collected Algorithms from ACM. Algorithm appeared in ACM-Trans. Math. Software, vol.8, no. 2, Jun., 1982, p. 190.

When implementing the skyline data structure, we can go back to the `csr` format, for the pupose of pretty-printing:

```
cout << ps << color << logscale << csr<Float>(m);
```

Implementation

```
template<class T>
class ssk : smart_pointer<spooles_rep<T> > {
public:
// typedefs:

        typedef          T                           element_type;
        typedef typename spooles_rep<T>::size_type   size_type;

// allocators/deallocators:

        ssk ();
        explicit ssk<T> (const csr<T>&);

// accessors:

        size_type nrow () const;
        size_type ncol () const;
        size_type nnz  () const;

// factorisation and solver:

        void solve(const vec<T>& b, vec<T>& x) const;
        vec<T> solve(const vec<T>& b) const;
        void factorize_ldlt();
protected:
        const spooles_rep<T>& data() const {
                return smart_pointer<spooles_rep<T> >::data();
        }
        spooles_rep<T>& data() {
                return smart_pointer<spooles_rep<T> >::data();
        }
};
template <class T>
ssk<T> ldlt (const csr<T>& m);
```

Implementation

```
template<class T>
class ssk : smart_pointer<umfpack_rep<T> > {
public:
// typedefs:

        typedef          T                          element_type;
        typedef typename umfpack_rep<T>::size_type  size_type;

// allocators/deallocators:

        ssk ();
        explicit ssk<T> (const csr<T>&);

// accessors:

        size_type nrow () const;
        size_type ncol () const;
        size_type nnz  () const;

// factorisation and solver:

        void solve(const vec<T>& b, vec<T>& x) const;
        vec<T> solve(const vec<T>& b) const;
        void factorize_ldlt();
        void factorize_lu();
protected:
        const umfpack_rep<T>& data() const {
                return smart_pointer<umfpack_rep<T> >::data();
        }
        umfpack_rep<T>& data() {
                return smart_pointer<umfpack_rep<T> >::data();
        }
};
template <class T>
ssk<T> ldlt (const csr<T>& m);
```

5.17 `basic_diag` - diagonal matrix

(Source file: '`skit/lib/diag.h`')

Description

The class implements a diagonal matrix. A declaration whithout any parametrers correspond to a null size matrix:

```
        basic_diag<Float> d;
```

The constructor can be invocated whith a size parameter:

```
basic_diag<Float> d(n);
```

or an initialiser, either a vector (see Section 5.14 [vec class], page 68):

```
basic_diag<Float> d = basic_diag(v);
```

or a csr matrix see Section 5.15 [csr class], page 70:

```
basic_diag<Float> d = basic_diag(a);
```

The conversion from `basic_diag` 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 *nrow* 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.

Note

Since the `basic_diag` class derives from the `vec`, the `basic_diag` class present also a STL-like interface.

Implementation

```
template<class T>
class basic_diag : public vec<T> {
public:

// typedefs:

    typedef typename vec<T>::element_type element_type;
    typedef typename vec<T>::size_type    size_type;
    typedef typename vec<T>::iterator     iterator;

// allocators/deallocators:

    explicit basic_diag (size_type sz = 0);
    explicit basic_diag (const vec<T>& u);
    explicit basic_diag (const csr<T>& a);

// assignment:

    basic_diag<T> operator = (const T& lambda);

// accessors:

    size_type nrow () const { return vec<T>::size(); }
    size_type ncol () const { return vec<T>::size(); }

// basic_diag as a preconditionner: solves D.x=b
```

```
    vec<T> solve (const vec<T>& b) const;
    vec<T> trans_solve (const vec<T>& b) const;
};
template <class T>
basic_diag<T> dcat (const basic_diag<T>& a1, const basic_diag<T>& a2);

template <class T>
basic_diag<T> operator / (const T& lambda, const basic_diag<T>& d);

template <class T>
vec<T>
operator * (const basic_diag<T>& d, const vec<T>& x);

template<class T>
vec<T> left_div (const vec<T>& x, const basic_diag<T>& d);
```

5.18 `permutation` – permutation matrix

(Source file: '`skit/lib/permutation.h`')

Description

This class implements a permutation matrix. Permutation matrix are used in factorization implementation for optimizing the skyline format, as used by the implementation of the `ssk` class (see Section 5.16 [ssk class], page 72).

Let a be a square invertible matrix in `csr` format (see Section 5.15 [csr class], page 70):

```
csr<double> a;
```

We get the optimal Gibbs permutation matrix using Gibbs, Pooles and Stockmeyer renumbering algorithm by:

```
permutation p = gibbs(a);
```

Implementation

```
class permutation : public Array<std::vector<int>::size_type> {
public:

// typedefs:

    typedef Array<int>::size_type       size_type;
    typedef Array<int>::difference_type difference_type;

// allocators/deallocators:

    explicit permutation (size_type n = 0);
};
template<class T>
permutation gibbs (const csr<T>& a);
```

5.19 `ic0` - incomplete Choleski factorization

(Source file: '`skit/lib/ic0.h`')

Description

The class implements the icomplete Choleski factorization IC0(A) when A is a square symmetric matrix stored in CSR format.

```
csr<double> a = ...;
ic0<double> fact(a);
```

Implementation

```
template <class T>
class basic_ic0 : public csr<T> {
public:
  typedef typename csr<T>::size_type size_type;
  typedef          T                 element_type;
// constructors:
  basic_ic0 ();
  explicit basic_ic0 (const csr<T>& a);
// solver:
  void inplace_solve (vec<T>& x) const;
  void solve (const vec<T>& b, vec<T>& x) const;
  vec<T> solve (const vec<T>& b) const;
// accessors:
  size_type nrow () const { return csr<T>::nrow(); }
  size_type ncol () const { return csr<T>::ncol(); }
  size_type nnz  () const { return csr<T>::nnz(); }
};
template <class T>
basic_ic0<T> ic0 (const csr<T>& a);
```

5.20 `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);
};
```

5.21 `iorheo` - input and output functions and manipulation

(Source file: '`util/lib/iorheo.h`')

Example

input geo in standard file format:

```
cin >> g;
```

output geo in standard file format:

```
cout << g;
```

output geo in gnuplot format:

```
cout << gnuplot << g;
```

5.22 `catchmark` - iostream manipulator

(Source file: '`util/lib/catchmark.h`')

Description

The `catchmark` is used for building labels used for input-output of vector-valued fields (see Section 5.5 [field class], page 54):

```
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);
        friend std::istream& operator >> (std::istream& is, const catchmark& m);
        friend std::ostream& operator << (std::ostream& os, const catchmark& m);
    protected:
        std::string _mark;
};
```

5.23 `irheostream`, `orheostream` - large data streams

(Source file: '`util/lib/rheostream.h`')

Abstract

This class provides a stream interface for large data management. File decompresion is assumed using `gzip` and a recursive seach 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 added, and compression is done with gzip, adding an additional '`.gz`' suffix.

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 mentionned 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`' exits, 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 mentionned 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 Section 5.21 [iorheo class], page 79).

Implementation

```
class irheostream : public io::filtering_stream<io::input> {
public:
    irheostream() : io::filtering_stream<io::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;
};
class orheostream : public io::filtering_stream<io::output> {
public:
    orheostream() : io::filtering_stream<io::output>() {}
    orheostream(const std::string& name, const std::string& suffix = std::string())
    virtual ~orheostream();
    void open  (const std::string& name, const std::string& suffix = std::string())
    void close();
protected:
    std::ofstream _ofs;
};
std::string itos (std::string::size_type i);
std::string ftos (const Float& x);

// catch first occurence of string in file
bool scatch (std::istream& in, const std::string& ch);

// 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);

// "/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&);
```

6 Algorithms

6.1 `riesz_representer` - integrate a function by using quadrature formulae

(Source file: '`nfem/lib/riesz_representer.h`')

Description

The function `riesz_representer` implements the approximation of an integral by using quadrature formulae. This is an expreimental implementation: please do not use yet for practical usage.

Synopsys

template <class Function> field riesz_representer (const space& Vh, const Function& f);

template <class Function> field riesz_representer (const space& Vh, const Function& f, quadrature_option_type qopt);

Example

The following code compute the Riesz representant, denoted by `mfh` of f(x), and the integral of f over the domain omega:

```
Float f(const point& x);
...
space Vh (omega_h, "P1");
field mfh = riesz_representer(Vh, f);
Float int_f = dot(mfh, field(Vh,1.0));
```

The Riesz representer is the `mfh` vector of values:

```
mfh(i) = integrate f(x) phi_i(x) dx
```

where phi_i is the i-th basis function in Vh and the integral is evaluated by using a quadrature formulae. By default the quadrature formule is the Gauss one with the order equal to the polynomial order of Vh. Alternative quadrature formulae and order is available by passing an optional variable to riesz_representer.

Implementation

```
template <class Function>
field
riesz_representer (
    const space& Vh,
    const Function& f,
    quadrature_option_type qopt
       = quadrature_option_type(quadrature_option_type::max_family,0))
```

6.2 newton – Newton nonlinear algorithm

(Source file: 'nfem/lib/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:
  my_problem();
  field residue (const field& uh) const;
  void update_derivative (const field& uh) const;
  field derivative_solve (const field& mrh) const;
  Float norm (const field& uh) const;
  Float dual_norm (const field& Muh) const;
};
```

See the example p-laplacian.h in the user's documentation for more.

Implementation

```
template <class Problem, class Field>
int newton (Problem P, Field& uh, Float& tol, size_t& max_iter, std::ostream *p_cer
    if (p_cerr) *p_cerr << "# Newton: n r" << std::endl;
    for (size_t n = 0; true; n++) {
      Field rh = P.residue(uh);
      Float r = P.dual_norm(rh);
      if (p_cerr) *p_cerr << n << " " << r << std::endl;
      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;
    }
}
```

6.3 damped_newton – damped Newton nonlinear algorithm

(Source file: 'nfem/lib/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);
```

The `my_problem` class may contains methods for the evaluation of F (aka residue) and its derivative:

```
class my_problem {
public:
  my_problem();
  field residue (const field& uh) const;
  void update_derivative (const field& uh) const;
  field derivative_trans_mult (const field& mrh) const;
  field derivative_solve (const field& mrh) const;
  Float norm (const field& uh) const;
  Float dual_norm (const field& Muh) const;
};
```

See the example `p-laplacian.h` in the user's documentation for more.

Implementation

```
template <class Problem, class Field, class Real, class Size>
int damped_newton (Problem P, Field& u, Real& tol, Size& max_iter, std::ostream* p_
  return damped_newton(P, newton_identity_preconditioner(), u, tol, max_iter, p_cer
}
```

6.4 pcg – conjugate gradient algorithm.

(Source file: '`skit/lib/pcg.h`')

Synopsis

```
template <class Matrix, class Vector, class Preconditioner, class Real>
int pcg (const Matrix &A, Vector &x, const Vector &b,
  const Preconditioner &M, int &max_iter, Real &tol, std::ostream *p_cerr=0);
```

Example

The simplest call to 'pcg' has the folling form:

```
size_t max_iter = 100;
double tol = 1e-7;
int status = pcg(a, x, b, EYE, max_iter, tol, &cerr);
```

Description

pcg 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
max_iter	the number of iterations performed before the tolerance was reached
tol	the residual after the final iteration

Note

pcg is an iterative template routine.

pcg 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/i

Implementation

```
template < class Matrix, class Vector, class Preconditioner, class Real, class Size
int pcg(const Matrix &A, Vector &x, const Vector &Mb, const Preconditioner &M,
        Size &max_iter, Real &tol, std::ostream *p_cerr = 0, std::string label = "c
{
    Vector b = M.solve(Mb);
    Real norm2_b = dot(Mb,b);
    if (norm2_b == Real(0)) norm2_b = 1;
    Vector Mr = Mb - A*x;
    Real  norm2_r = 0;
    if (p_cerr) (*p_cerr) << "[" << label << "] #iteration residue" << std::endl;
    Vector p;
    for (Size n = 0; n <= max_iter; n++) {
        Vector r = M.solve(Mr);
        Real prev_norm2_r = norm2_r;
        norm2_r = dot(Mr, r);
        if (p_cerr) (*p_cerr) << "[" << label << "] " << n << " " << ::sqrt(norm2_r
        if (norm2_r <= sqr(tol)*norm2_b) {
          tol = ::sqrt(norm2_r/norm2_b);
          max_iter = n;
          return 0;
```

```
            }
            if (n == 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;
        }
        tol = ::sqrt(norm2_r/norm2_b);
        return 1;
    }
```

6.5 pminres – conjugate gradient algorithm.

(Source file: 'skit/lib/pminres.h')

Synopsis

```
template <class Matrix, class Vector, class Preconditioner, class Real>
int pminres (const Matrix &A, Vector &x, const Vector &b, const Preconditioner &
  int &max_iter, Real &tol, std::ostream *p_cerr=0);
```

Example

The simplest call to 'pminres' has the folling form:

```
size_t max_iter = 100;
double tol = 1e-7;
int status = pminres(a, x, b, EYE, max_iter, tol, &cerr);
```

Description

`pminres` 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
`max_iter`	the number of iterations performed before the tolerance was reached
`tol`	the residual after the final iteration

Note

`pminres` 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/iml++`.

Implementation

```
template <class Matrix, class Vector, class Preconditioner, class Real, class Size>
int pminres(const Matrix &A, Vector &x, const Vector &Mb, const Preconditioner &M,
  Size &max_iter, Real &tol, std::ostream *p_cerr = 0, std::string label = "minres")
{
  Vector b = M.solve(Mb);
  Real norm_b = sqrt(fabs(dot(Mb,b)));
  if (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));
  if (p_cerr) (*p_cerr) << "[" << label << "] #iteration residue" << std::endl;
  if (p_cerr) (*p_cerr) << "[" << label << "] 0 " << norm_r/norm_b << std::endl;
  if (beta2 < 0 || norm_r <= tol*norm_b) {
    tol = norm_r/norm_b;
    max_iter = 0;
    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.size(), 0.);
  Vector Mv_old (x.size(), 0.);
  Vector w      (x.size(), 0.);
  Vector w_old  (x.size(), 0.);
  Vector w_old2 (x.size(), 0.);
  for (Size n = 1; n <= max_iter; n++) {
    // 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) {
        warning_macro ("pminres: machine precision problem");
        tol = norm_r/norm_b;
        max_iter = n;
        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;
    if (p_cerr) (*p_cerr) << "[" << label << "] " << n << " " << norm_r/norm_b << s
    if (norm_r <= tol*norm_b) {
      tol = norm_r/norm_b;
      max_iter = n;
      return 0;
    }
  }
  tol = norm_r/norm_b;
  return 1;
}
```

6.6 puzawa – Uzawa algorithm.

(Source file: 'skit/lib/puzawa.h')

Synopsis

```
template <class Matrix, class Vector, class Preconditioner, class Real>
int puzawa (const Matrix &A, Vector &x, const Vector &b, const Preconditioner &
  int &max_iter, Real &tol, const Real& rho, std::ostream *p_cerr=0);
```

Example

The simplest call to 'puzawa' has the folling form:

```
size_t max_iter = 100;
double tol = 1e-7;
int status = puzawa(A, x, b, EYE, max_iter, tol, 1.0, &cerr);
```

Description

puzawa solves the linear system A*x=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 'pcg' that computes automatically the optimal descdnt 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
max_iter	the number of iterations performed before the tolerance was reached
tol	the residual after the final iteration

Implementation

```
template < class Matrix, class Vector, class Preconditioner, class Real, class Size
int puzawa(const Matrix &A, Vector &x, const Vector &Mb, const Preconditioner &M,
    Size &max_iter, Real &tol, const Real& rho,
    std::ostream *p_cerr, std::string label)
{
    Vector b = M.solve(Mb);
    Real norm2_b = dot(Mb,b);
    Real norm2_r = norm2_b;
    if (norm2_b == Real(0)) norm2_b = 1;
    if (p_cerr) (*p_cerr) << "[" << label << "] #iteration residue" << std::endl;
    for (Size n = 0; n <= max_iter; n++) {
        Vector Mr = A*x - Mb;
        Vector r = M.solve(Mr);
```

```
            norm2_r = dot(Mr, r);
            if (p_cerr) (*p_cerr) << "[" << label << "] " << n << " " << sqrt(norm2_r/n
            if (norm2_r <= sqr(tol)*norm2_b) {
              tol = sqrt(norm2_r/norm2_b);
              max_iter = n;
              return 0;
            }
            x  -= rho*r;
        }
        tol = sqrt(norm2_r/norm2_b);
        return 1;
    }
```

6.7 pcg_abtb, pcg_abtbc, pminres_abtb, pminres_abtbc — solvers for mixed linear problems

(Source file: '`skit/lib/mixed_solver.h`')

Synopsis

```
template <class Matrix, class Vector, class Solver, class Preconditioner, class
int pcg_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, Size& max_iter, Real& tol,
  std::ostream *p_cerr = 0, std::string label = "pcg_abtb");

template <class Matrix, class Vector, class Solver, class Preconditioner, class
int pcg_abtbc (const Matrix& A, const Matrix& B, const Matrix& C, Vector& u, Ve
  const Vector& Mf, const Vector& Mg, const Preconditioner& S1,
  const Solver& inner_solver_A, Size& max_iter, Real& tol,
  std::ostream *p_cerr = 0, std::string label = "pcg_abtbc");
```

The synopsis is the same with the pminres 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:

```
[ A  B^T ] [ u ]    [ Mf ]
[        ] [   ]  = [    ]
[ B  -C  ] [ p ]    [ Mg ]
```

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. Formaly u = inv(A)*(Mf - B^T*p) and the reduced system writes for all non-singular matrix S1:

```
inv(S1)*(B*inv(A)*B^T)*p = inv(S1)*(B*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*inv(A)*B^T. Both direct or iterative solvers for S1*q = t are supported. Application of 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 stoping 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.

6.8 `bicgstab` - bi-conjugate gradient stabilized method

(Source file: '`skit/lib/bicgstab.h`')

Synopsis

```
int bicgstab (const Matrix &A, Vector &x, const Vector &b,
              const Preconditioner &M, int &max_iter, Real &tol);
```

Example

The simplest call to 'bicgstab' has the folling form:

```
int status = bicgstab(a, x, b, EYE, 100, 1e-7);
```

Description

`bicgstab` solves the unsymmetric linear system Ax = b using the preconditioned bi-conjugate gradient stabilized 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
`max_iter`	the number of iterations performed before the tolerance was reached
`tol`	the residual after the final iteration

Note

`bicgstab` is an iterative template routine.

`bicgstab` follows the algorithm described on p. 24 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/i`

6.9 `gmres` – generalized minimum residual method

(Source file: '`skit/lib/gmres.h`')

Synopsis

```
template <class Operator, class Vector, class Preconditioner,
    class Matrix, class Real, class Int>
int gmres (const Operator &A, Vector &x, const Vector &b,
    const Preconditioner &M, Matrix &H, Int m, Int &max_iter, Real &tol);
```

Example

The simplest call to `gmres` has the folling form:

```
int m = 6;
dns H(m+1,m+1);
int status = gmres(a, x, b, EYE, H, m, 100, 1e-7);
```

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
`max_iter`	the number of iterations performed before the tolerance was reached
`tol`	the residual after the final iteration

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 though 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 acess.

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/i`

Implementation

```
template < class Matrix, class Vector, class Int >
void
Update(Vector &x, Int k, Matrix &h, Vector &s, Vector v[])
{
  Vector y(s);

  // Backsolve:
  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 (Int j = 0; j <= k; j++)
    x += v[j] * y(j);
}

#ifdef TO_CLEAN
template < class Real >
Real
abs(Real x)
{
  return (x > Real(0) ? x : -x);
}
#endif // TO_CLEAN


template < class Operator, class Vector, class Preconditioner,
           class Matrix, class Real, class Int >
int
gmres(const Operator &A, Vector &x, const Vector &b,
      const Preconditioner &M, Matrix &H, const Int &m, Int &max_iter,
```

```
      Real &tol)
{
  Real resid;
  Int i, j = 1, k;
  Vector s(m+1), cs(m+1), sn(m+1), w;

  Real normb = norm(M.solve(b));
  Vector r = M.solve(b - A * x);
  Real beta = norm(r);

  if (normb == Real(0))
    normb = 1;

  if ((resid = norm(r) / normb) <= tol) {
    tol = resid;
    max_iter = 0;
    return 0;
  }

  Vector *v = new Vector[m+1];

  while (j <= max_iter) {
    v[0] = r * (1.0 / beta);    // ??? r / beta
    s = 0.0;
    s(0) = beta;

    for (i = 0; i < m && j <= max_iter; i++, j++) {
      w = M.solve(A * v[i]);
      for (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 (k = 0; k < i; k++)
        ApplyPlaneRotation(H(k,i), H(k+1,i), cs(k), sn(k));

      GeneratePlaneRotation(H(i,i), H(i+1,i), cs(i), sn(i));
      ApplyPlaneRotation(H(i,i), H(i+1,i), cs(i), sn(i));
      ApplyPlaneRotation(s(i), s(i+1), cs(i), sn(i));

      if ((resid = abs(s(i+1)) / normb) < tol) {
        Update(x, i, H, s, v);
        tol = resid;
        max_iter = j;
        delete [] v;
        return 0;
      }
```

```
    }
    Update(x, m - 1, H, s, v);
    r = M.solve(b - A * x);
    beta = norm(r);
    if ((resid = beta / normb) < tol) {
      tol = resid;
      max_iter = j;
      delete [] v;
      return 0;
    }
  }

  tol = resid;
  delete [] v;
  return 1;
}
template<class Real>
void GeneratePlaneRotation(Real &dx, 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 ApplyPlaneRotation(Real &dx, Real &dy, Real &cs, Real &sn)
{
  Real temp  =  cs * dx + sn * dy;
  dy = -sn * dx + cs * dy;
  dx = temp;
}
```

6.10 qmr – quasi-minimal residual algoritm

(Source file: ‘skit/lib/qmr.h’)

Synopsis

```
template <class Matrix, class Vector, class Preconditioner1,
    class Preconditioner2, class Real>
int qmr (const Matrix &A, Vector &x, const Vector &b,
```

```
const Preconditioner1 &M1, const Preconditioner2 &M2,
int &max_iter, Real &tol);
```

Example

The simplest call to 'qmr' has the folling form:

```
int status = qmr(a, x, b, EYE, EYE, 100, 1e-7);
```

Description

`qmr` solves the unsymmetric linear system Ax = b using the the quasi-minimal 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
`max_iter`	the number of iterations performed before the tolerance was reached
`tol`	the residual after the final iteration

A return value of 1 indicates that the method did not reach the specified convergence tolerance in the maximum numbefr of iterations. A return value of 2 indicates that a breackdown associated with `rho` occurred. A return value of 3 indicates that a breackdown associated with `beta` occurred. A return value of 4 indicates that a breackdown associated with `gamma` occurred. A return value of 5 indicates that a breackdown associated with `delta` occurred. A return value of 6 indicates that a breackdown associated with `epsilon` occurred. A return value of 7 indicates that a breackdown associated with `xi` occurred.

Note

`qmr` is an iterative template routine.

`qmr` follows the algorithm described on p. 24 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/i`

7 Forms

7.1 d_dx*i* – derivatives

(Source file: 'nfem/form_element/d_dx0.h')

Synopsis

```
form (const space V, const space& M, "d_dx0");
form (const space V, const space& M, "d_dx1");
form (const space V, const space& M, "d_dx2");
```

Description

Assembly the form associated to a derivative operator from the V finite element space to the M one: $$ b_i(u,q) = \int_\Omega \partial u \over \partial x_i \, q \, dx, \ \ i\in \ 0,1,2 \ $$ In the axisymetric rz case, the form is defined by $$ b_0(u,q) = \int_\Omega \partial u \over \partial r \, q \, r dr dz, $$ If the V space is a P1 (resp. P2) finite element space, the M space may be a P0 (resp. P1d) one.

Example

The following piece of code build the Laplacian form associated to the P1 approximation:

```
geo omega ("square");
space V (omega, "P1");
space M (omega, "P0");
form  b (V, M, "d_dx0");
```

Limitations

Only edge, triangular and tetrahedal finite element meshes are yet supported.

7.2 mass – L2 scalar product

(Source file: 'nfem/form_element/mass.h')

Synopsis

```
form(const space& V, const space& V, "mass");
form(const space& M, const space& V, "mass");
form (const space& V, const space& V, "mass", const domain& gamma);
form_diag(const space& V, "mass");
```

Description

Assembly the matrix associated to the L2 scalar product of the finite element space V. $$ m(u,v) = \int_\Omega u v \, dx $$

The V space may be either a `P0`, `P1`, `P2`, `bubble`, `P1d` and `P1d` finite element spaces for building a form see Section 5.7 [form class], page 56.

The use of quadrature formulae is sometime usefull for building diagonal matrix. These approximate matrix are eay to invert. This procedure is available for `P0` and `P1` approximations.

Notes that when dealing with discontinuous finite element space, i.e. `P0` and `P1d`, the corresponding mass matrix is block diagonal, and the `inv_mass` form may be usefull.

When two different space M and V are supplied, assembly the matrix associated to the projection operator from one finite element space M to space V. $$ m(q,v) = \int_\Omega q v \, dx $$ for all $q \in M$ and $v \in V$.

This form is usefull for instance to convert discontinuous gradient components to a continuous approximation. The transpose operator may also be usefull to performs the opposite operation.

The following V and M space approximation combinations are supported for the `mass` form: P0-P1, P0-P1d, P1d-P2, P1-P1d and P1-P2.

Example

The following piece of code build the mass matrix associated to the P1 approximation:

```
geo g("square");
space V(g, "P1");
form m(V, V, "mass");
```

The use of lumped mass form write also:

```
form_diag md(V, "mass");
```

The following piece of code build the projection form:

```
geo g("square");
space V(g, "P1");
space M(g, "P0");
form m(M, V, "mass");
```

Scalar product on the boundary

Assembly the matrix associated to the L2 scalar product related to a boundary domain of a mesh and a specified polynomial approximation. These forms are usefull when defining non-homogeneous Neumann or Robin boundary conditions.

Let W be a space of functions defined on Gamma, a subset of the boundary of the whole domain Omega. $$ m(u,v) = \int_\Gamma u \, v \, ds, \ \ \forall u, v \in W $$ Let V a space of functions defined on Omega and gamma the trace operator from V into W. $$ mb(u,v) = \int_\Gamma u \, \gamma v \, ds, \ \ \forall u \in W, v \in V $$ $$ ab(u,v) = \int_\Gamma \gamma u \, \gamma v \, ds, \ \ \forall u, v \in V $$

Example

The following piece of code build forms for the P1 approximation, assuming that the mesh contains a domain named `boundary`:

```
geo omega ("square");
domain gamma = omega.boundary();
space V  (omega, "P1");
space W  (omega, gamma, "P1");
form  m  (W, W, "mass");
form  mb (W, V, "mass");
form  ab (V, V, "mass", gamma);
```

7.3 `inv_mass` – invert of L2 scalar product

(Source file: '`nfem/form_element/inv_mass.h`')

Synopsis

```
form(const space& V, const space& V, "inv_mass");
```

Description

Assembly the invert of the matrix associated to the L2 scalar product of the finite element space V: $$ m(u,v) = \int_\Omega u v \, dx $$ The V space may be either a `P0` or `P1d` discontinuous finite element spaces see Section 5.7 [form class], page 56.

Example

The following piece of code build the invert of the mass matrix associated to the `P1d` approximation:

```
geo omega_h ("square");
space Vh (omega_h , "P1d");
form im (Vh, Vh, "inv_mass");
```

7.4 `grad_grad` – -Laplacian operator

(Source file: '`nfem/form_element/grad_grad.h`')

Synopsis

```
form(const space V, const space& V, "grad_grad");
```

Description

Assembly the form associated to the Laplacian operator on a finite element space `V`: $$ a(u,v) = \int_\Omega \nabla u . \nabla v \, dx $$ The V space may be a either `P1` , `P2` or `P1d` finite element space. See also Section 5.7 [form class], page 56 and Section 5.4 [space class], page 49.

Example

The following piece of code build the Laplacian form associated to the `P1` approximation:

```
geo g("square");
space V(g, "P1");
form a(V, V, "grad_grad");
```

7.5 `d_ds` – Curvilinear derivative

(Source file: '`nfem/form_element/d_ds.h`')

Synopsis

```
form(const space& V, const space& W, "d_ds");
```

Description

Assembly the matrix associated to the following integral: $$ m(u,v) = \int_\Gamma du/ds v \, ds $$

7.6 `d2_ds2` – Curvilinear second derivative

(Source file: '`nfem/form_element/d2_ds2.h`')

Synopsis

```
form(const space& V, const space& W, "d2_ds2");
```

Description

Assembly the matrix associated to the following integral: $$ m(u,v) = \int_\Gamma d^2 u/ds^2 v \, ds $$

7.7 `grad` – gradient operator

(Source file: '`nfem/form_element/grad.h`')

Synopsis

```
form(const space V, const space& M, "grad");
```

Description

Assembly the form associated to the gradient operator on a finite element space `V`: $$ b(u, \bf q) = \int_\Omega \nabla u . \bf q \, dx $$ The V space may be a either `P1` or `P2` finite element space, while the M space may be `P0` or `P1d` respectively. See also Section 5.7 [form class], page 56 and Section 5.4 [space class], page 49.

Example

The following piece of code build the divergence form associated to the `P1` approximation:

```
geo omega("square");
space V(omega, "P1");
space M(omega, "P0", "vector");
form b(V, M, "grad");
```

7.8 `curl` – curl operator

(Source file: '`nfem/form_element/curl.h`')

Synopsis

```
form(const space V, const space& M, "curl");
```

Description

Assembly the form associated to the curl operator on finite element space. In three dimensions, both V and M are vector-valued: $$ b(\bf u,\bf q) = \int_\Omega \bf curl\,\bf u . \bf q \, dx $$ In two dimensions, only V is vector-valued. The V space may be a either `P2` finite element space, while the M space may be `P1d`. See also Section 5.7 [form class], page 56 and Section 5.4 [space class], page 49.

Example

The following piece of code build the divergence form associated to the `P2` approximation for a three dimensional geometry:

```
geo omega("cube");
space V(omega, "P2",  "vector");
space M(omega, "P1d", "vector");
form b(V, M, "curl");
```

while this code becomes in two dimension:

```
geo omega("square");
space V(omega, "P2",  "vector");
space M(omega, "P1d");
form b(V, M, "curl");
```

7.9 `div` – divergence operator

(Source file: '`nfem/form_element/div.h`')

Synopsis

```
form(const space V, const space& M, "div");
```

Description

Assembly the form associated to the divergence operator on a finite element space `V`: $$ b(\bf u,q) = \int_\Omega \rm div\,\bf u \, q \, dx $$ The V space may be a either `P1` or `P2` finite element space, while the M space may be `P0` or `P1d` respectively. See also Section 5.7 [form class], page 56 and Section 5.4 [space class], page 49.

Example

The following piece of code build the divergence form associated to the `P1` approximation:

```
geo omega("square");
space V(omega, "P1", "vector");
space M(omega, "P0");
form b(V, M, "div");
```

7.10 `2D` – rate of deformation tensor

(Source file: '`nfem/form_element/2D.h`')

Synopsis

```
form (const space V, const space& T, "2D");
```

Description

Assembly the form associated to the rate of deformation tensor, i.e. the symmetric part of the gradient of a vector field. These derivative are usefull in fluid mechanic and elasticity. $$ b(\bf u,\tau) = \int_\Omega 2D(\bf u) : \, \tau \, dx, \forall u \in V \ \rm and \ q \in T. $$ where $$ 2 D(\bf u) = \nabla \bf u + \nabla \bf u^T $$

If the V space is a vector-valued `P1` (resp. `P2`) finite element space, the T space may be a tensor-valued `P0` (resp. `P1d`) one.

Example

The following piece of code build the Laplacian form associated to the P1 approximation:

```
geo omega ("square");
space V (omega, "P1", "vector");
space T (omega, "P0", "tensor");
form  b (V, T, "2D");
```

7.11 2W – vorticity tensor

(Source file: '`nfem/form_element/2W.h`')

Synopsis

```
form (const space V, const space& T, "2W");
```

Description

Assembly the form associated to the vorticity tensor, i.e. the unsymmetric part of the gradient of a vector field. These derivative are usefull in fluid mechanic. $$ b(\bf u,\tau) = \int_\Omega 2W(\bf u) : \, \tau \, dx, \forall u \in V \ \rm and \ q \in T. $$ where $$ 2 D(\bf u) = \nabla \bf u - \nabla \bf u^T $$ If the V space is a vector-valued `P1` (resp. `P2`) finite element space, the T space may be a tensor-valued `P0` (resp. `P1d`) one.

Example

The following piece of code build the Laplacian form associated to the P1 approximation:

```
geo omega ("square");
space V (omega, "P1", "vector");
space T (omega, "P0", "tensor");
form  b (V, T, "2W");
```

7.12 2D_D – -div 2D operator

(Source file: 'nfem/form_element/2D_D.h')

Synopsis

```
form (const space V, const space& V, "2D_D");
```

Description

Assembly the form associated to the $\bf div(2D(.))$ components of the operator on the finite element space V: $$ a(\bf u,\bf v) = \int_\Omega 2 D_ij(\bf u) . D_ij(\bf v) \, dx, \ \ 0 \leq i \leq j < 3 $$ where $$ D_ij(u) = 1 \over 2 \left(\partial u_i \over \partial x_j + \partial u_j \over \partial x_i \right) $$ This form is usefull when considering elasticity or Stokes problems.

Example

Here is an example of the vector-valued form:

```
geo omega ("square");
space V (omega, "P2", "vector");
form  a (V, V, "2D_D");
```

Note that a factor two is here applied to the form. This factor is commonly used in practice.

7.13 div_div – -grad div operator

(Source file: 'nfem/form_element/div_div.h')

Synopsis

```
form (const space V, const space& V, "div_div");
```

Description

Assembly the form associated to the `grad(div(.))` operator on the finite element space `V`: $$ a(\bf u,\bf v) = \int_\Omega \rm div(\bf u) \rm div(\bf v) \, dx $$ This form is usefull when considering elasticity problem.

Example

Here is an example of the vector-valued form:

```
geo omega ("square");
space V (omega, "P2", "vector");
form  a (V, V, "div_div");
```

7.14 `convect` – discontinuous Galerkin

(Source file: '`nfem/form_element/convect.h`')

Synopsis

```
form(const space& V, const space& V, "convect", uh);
```

Description

Assembly the matrix associated to the discontinuous Galerkin method on the finite element space V. $$ c(u,v) = \int_\Omega a.grad(u) v \, dx + skips... $$ for all $u,v \in V$, where the vector field a is given. The V space may `P1d` finite element spaces for building the form.

8 Internals

8.1 `form_element` - bilinear form on a single element

(Source file: '`nfem/lib/form_element.h`')

Synopsys

The `form_element` class defines functions that compute a bilinear form defined between two polynomial basis on a single geometrical element. This bilinear form is represented by a matrix.

The bilinear form is designated by a string, e.g. "mass", "grad_grad", ... indicating the form. The form depends also of the geometrical element: triangle, square, tetrahedron (see Section 8.9 [geo_element internal], page 125).

Implementation note

The `form_element` class is managed by (see Section 8.17 [smart_pointer internal], page 133). This class uses a pointer on a pure virtual class `form_element_rep` while the effective code refers to the specific concrete derived classes: mass, grad_grad, etc.

Implementation

```
class form_element : public smart_pointer<form_element_rep> {
public:

// typedefs:

    typedef size_t size_type;

// allocators:

    form_element (std::string name = "");

// modifier:

    void initialize (const space& X, const space& Y) const;

    //  optional, for scalar-weighted forms:
    void set_weight (const field& wh) const;

        //  for special forms for which coordinate-system specific weights (ie, cyl
        //  coordinates weights) should not be used:
        void set_use_coordinate_system_weight(bool use) const;

// accessor:

    void operator() (const geo_element& K, ublas::matrix<Float>& m) const;
```

```
    const field& get_weight() const;
};
```

8.2 geomap - discrete mesh advection by a field: gh(x)=fh(x-dt*uh

(Source file: 'nfem/lib/geomap.h')

Synopsys

The class geomap is a fundamental class used for the correspondance between fields defined on different meshes or for advection problems. This class is used for the method of characteristic.

Example

The following code compute gh(x)=fh(x-dt*uh(x)):

```
field uh = interpolate (Vh, u);
field fh = interpolate (Fh, f);
geomap X (Fh, -dt*uh);
field gh = compose (fh, X);
```

For a complete example, see 'convect.cc' in the example directory.

Implementation

```
struct geomap_option_type {
    size_t n_track_step; // loop while tracking: y = X_u(x)
    geomap_option_type() : n_track_step(1) {}
};
class geomap : public Vector<meshpoint> {
public:
        geomap () : advected(false), use_space(true) {}
        //  Maps a quadrature-dependent lattice over Th_1 onto Th_2 triangulation.
        geomap (const geo& Th_2, const geo& Th_1,
                        std::string quadrature="default", size_t order=0, bool allo

        //  Maps a quadrature-dependent lattice over Th_1 onto Th_2 triangulation
        //  thro' an offset vector field u to be defined on Th_1.
        geomap (const geo& Th_2, const geo& Th_1, const field& advection_h_1,
                        std::string quadrature="default", size_t order=0) ;

        //  TO BE REMOVED: backward compatibility
        //  Maps space Vh_1 dof's onto the meshpoints of Th_2 triangulation
        /*  geomap : ( Vh_1.dof's ) |--> ( Th_2 )
         */
        geomap (const geo& Th_2, const space& Vh_1,
                         bool allow_approximate_edges=true );
```

```
        //  Same but for P1d, using points inside the triangle to preserve disconti
        geomap (const geo& Th_2, const space& Vh_1,
                        Float barycentric_weight );

        //  TO BE REMOVED: backward compatibility
        //  Maps space Vh_1 dof's onto the meshpoints of Th_2 triangulation
        //  thro' an offset u to be defined on Vh_1 dof's.
        /*  geomap : ( Vh_1.dof's ) |--> ( Th_2 )
         */
        geomap (const geo& Th_2, const space& Vh_1, const field& advection_h_1);

        //  Does the same, but assumes that Th_2 is the triangulation for Vh_1
        geomap (const space& Vh_2, const field& advection_h_1, const geomap_option_

        ~geomap () {};

//accessors
        //  TO BE REMOVED: backward compatibility
        const space&
        get_space () const
                { if (!use_space) error_macro("Lattice-based geomaps use no space")
                  return _Vh_1;
                };

        const geo&
        get_origin_triangulation () const
                { return _Th_1; };

        const geo&
        get_target_triangulation () const
                { return _Th_2; };

        size_t
        order () const
                {
                //  TO BE REMOVED: backward compatibility
                if (!use_space)
                return _order;
                }

        const std::string&
        quadrature () const
                {
                //  TO BE REMOVED: backward compatibility
                if (!use_space)
                return _quadrature; }

        bool is_inside (size_t dof) const { return _is_inside [dof]; }
        bool no_barycentric_weight() const { return _barycentric_weight == Float(0)
```

```
        Float barycentric_weight() const { return _barycentric_weight; }

protected:
        friend class fieldog;
        //  use_space mode
        void init (const field& advection);
        //  non use_space mode
        void init ();
        meshpoint advect (const point& q, size_t iK_Th_1);
        meshpoint robust_advect_1 (const point& x0, const point& va, bool& is_inside
        meshpoint robust_advect_N (const point& x0, const point& v0, const field& vh


// data:
        geo _Th_1;
        geo _Th_2;
        field _advection;
        bool advected;
        std::string _quadrature;
        size_t _order;
        bool _allow_approximate_edges;

        // TO BE REMOVED:
        bool use_space;
        space _Vh_1;

        std::vector<point> quad_point;
        std::vector<Float> quad_weight;

        // flag when a dof go outside of the domain
        std::vector<bool> _is_inside;

        Float _barycentric_weight;
        geomap_option_type _option;
 };

inline
geomap::geomap (const geo& Th_2, const space& Vh_1, const field& advection) :
        _Th_1 (Vh_1.get_geo()), _Th_2 (Th_2), _advection(advection), advected(true)
        _quadrature("default"), _order(0),
        _allow_approximate_edges(true), use_space(true), _Vh_1(Vh_1),
        _is_inside(), _barycentric_weight(0)
                { init (advection); };

inline
geomap::geomap (const space& Vh_1, const field& advection, const geomap_option_type
        _Th_1 (Vh_1.get_geo()), _Th_2 (Vh_1.get_geo()), _advection(advection), adve
        _quadrature("default"), _order(0),
        _allow_approximate_edges(true), use_space(true), _Vh_1(Vh_1),
```

```
        _is_inside(), _barycentric_weight(0), _option(opt)
                { init (advection); };

inline
geomap::geomap (
    const geo& Th_2,
    const geo& Th_1,
    const field& advection,
    std::string quadrature,
    size_t order)
  :
    _Th_1 (Th_1),
    _Th_2 (Th_2),
    _advection(advection),
    advected(true),
    _quadrature(quadrature),
    _order(order),
    _allow_approximate_edges(true),
    use_space(false),
    _Vh_1(),
    _is_inside(),
    _barycentric_weight(0),
    _option()
{
    if (_advection.get_geo() !=_Th_1)
        error_macro("The advection field should be defined on the original mesh Th_
            << Th_1.name() << " and not " << _advection.get_geo().name());
    init();
}

inline
geomap::geomap (
    const geo& Th_2,
    const geo& Th_1,
    std::string quadrature,
    size_t order,
    bool allow_approximate_edges)
  : _Th_1 (Th_1),
    _Th_2 (Th_2),
    _advection(),
    advected(false),
    _quadrature(quadrature),
    _order(order),
    _allow_approximate_edges(allow_approximate_edges),
    use_space(false),
    _Vh_1(),
    _is_inside(),
    _barycentric_weight(0),
    _option()
```

```
{
    init();
}

//  Composition with a field
/*  f being a field of space V, X a geomap on this space, foX=compose(f,X) is their
 *  composition.
 */
field compose (const field& f, const geomap& g);
/*  send also a callback function when advection goes outside a domain
 *  this is usefull when using upstream boundary conditions
 */
field compose (const field& f, const geomap& g, Float (*f_outside)(const point&));
template <class Func>
field compose (const field& f, const geomap& g, Func f_outside);
```

Implementation

```
class fieldog: public field // and friend of geomap.
 {
public:
   // Constructor
        fieldog (const field& _f, const geomap& _g) : f(_f), g(_g) {};

        // Accessors
        Float
        operator() (const point& x) const;

        Float
        operator() (const meshpoint& x) const;

        Float
        operator() (const point& x_hat, size_t e) const
        { return operator() (meshpoint(x_hat,e)); };

        std::vector<Float>
        quadrature_values (size_t iK) const;

        std::vector<Float>
        quadrature_d_dxi_values (size_t i, size_t iK) const;

        const std::string&
        quadrature() const
        { return g._quadrature; };

        size_t
        order() const
        { return g._order; };
```

```
        const space&
        get_space() const
        { return f.get_space(); };

protected:
        field f;
        geomap g;
 };
```

8.3 `iofem` - input and output finite element manipulators

(Source file: '`nfem/lib/iofem.h`')

Description

This class implements some specific finite element manipulators. For a general presentation of stream manipulators, reports to class `iostream`.

Valuated manipulators

`origin`
`normal` set a cutting plane for visualizations and post-processing.

```
cout << cut << origin(point(0.5, 0.5, 0.5)) << normal(point(1
```

`topography`
specifies the topography field when representing a bidimensionnal field in tridimensionnal elevation.

```
cout << topography(zh) << uh;
```

This manipulator takes an agument that specifies a scalar field value `zh` for the elevation, while `uh` is the scalar field to represent. Then, the z-elevation takes zh(x,y)+uh(x,y).

8.4 `characteristic` - discrete mesh advection by a field: gh(x)=fh(x-dt*uh

(Source file: '`nfem/lib/characteristic.h`')

Synopsys

The class characteristic is a class implementing the Lagrange-Galerkin method. It is the extension of the method of characteristic in the finite element context. This is an expreimental implementation: please use the "geomap" one for practical usage.

Example

The following code compute the Riesz representant, denoted by "mgh" of gh(x)=fh(x-dt*uh(x)).

```
geo omega_h;
field uh = ...;
field fh = ...;
characteristic X (omega_h, -dt*uh);
field mgh = riesz_representer(Vh, compose(fh, X));
```

The Riesz representer is the "mgh" vector of values:

```
mgh(i) = integrate fh(x-dt*uh(x)) phi_i(x) dx
```

where phi_i is the i-th basis function in Vh and the integral is evaluated by using a quadrature formulae. By default the quadrature formule is Gauss-Lobatto with the order equal to the polynomial order of Vh. This quadrature formulae guaranties inconditional stability at any polynomial order (order 1: trapeze, order 2: simpson). Extension will accept in the future alternative quadrature formulae.

Implementation

```
class characteristic {
public:
        characteristic(const geo& bg_omega, const field& ah);
        const geo& get_background_geo() const;
        const field& get_advection() const;
protected:
        geo   _bg_omega;
        field _ah;
};
class field_o_characteristic {
public:
        field_o_characteristic(const field& fh, const characteristic& X);
        friend field_o_characteristic compose(
                const field& fh, const characteristic& X);
        Float operator() (const point& x) const;
        point vector_evaluate (const point& x) const;
        tensor tensor_evaluate (const point& x) const;
        const field& get_field() const;
        const geo& get_background_geo() const;
        const field& get_advection() const;
protected:
        field          _fh;
        characteristic _X;
        point advect (const point& x0) const;
};
```

8.5 `basis` - polynomial basis

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

Synopsys

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. The basis depends also of the reference element: triangle, square, tetrahedron (see Section 8.8 [reference_element internal], page 123). For instance, on a square, the "P1" string designates the common Q1 four-nodes basis on the reference square.

The nodes associated to the Lagrange polynomial basis are also available by its associated accessor.

Implementation note

The `basis` class is a see Section 8.17 [smart_pointer internal], page 133) class on a `basis_rep` class that is a pure virtual base class for effective bases, e.g. basis_P1, basis_P1, etc.

Implementation

```
class basis : public smart_pointer<basis_rep> {
public:

// typedefs:

    typedef size_t size_type;
    typedef basis_rep::dof_family_type dof_family_type;

// allocators:

    basis (std::string name = "");

// accessors:

    std::string name() const;
    size_type   degree() const;
    size_type   size (reference_element hat_K, dof_family_type family=reference_ele
    dof_family_type family() const;

    dof_family_type dof_family(
        reference_element hat_K,
        size_type         i_dof_local) const;

    Float eval(
        reference_element hat_K,
        size_type         i_dof_local,
        const point&      hat_x) const;
```

```
    point grad_eval(
        reference_element hat_K,
        size_type         i_dof_local,
        const point&      hat_x) const;

    basic_point<point> hessian_eval(
        reference_element hat_K,
        size_type         i_dof_local,
        const point&      hat_x) const;

    void eval(
        reference_element    hat_K,
        const point&         hat_x,
        std::vector<Float>&  values) const;

    void grad_eval(
        reference_element    hat_K,
        const point&         hat_x,
        std::vector<point>&  values) const;

    void hessian_eval(
        reference_element    hat_K,
        const point&         hat_x,
        std::vector<basic_point<point> >&  values) const;

    void hat_node(
        reference_element    hat_K,
        std::vector<point>&  hat_node) const;

    void dump(std::ostream& out = std::cerr) const;
};
```

8.6 quadrature - quadrature formulae on the reference lement

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

Synopsys

The quadrature class defines a container for a quadrature formulae on the reference element (see Section 8.8 [reference_element internal], page 123). 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.

$$\int_K p(x)\,dx = \sum_{i=0}^{n-1} w_i\, p(x_i)$$

```
| p(x) dx =  \    p(x_q) w_q
/ K          /__
             q=1
```

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

```
class quadrature {
public:

// typedefs:

    typedef quadrature_on_geo::size_type size_type;
    typedef quadrature_option_type::family_type family_type;
    typedef std::vector<weighted_point>::const_iterator const_iterator;

// allocators:

    quadrature (quadrature_option_type opt = quadrature_option_type());

// modifiers:

    void set_order (size_type order);
    void set_family (family_type ft);

// accessors:

    size_type      get_order() const;
    family_type    get_family() const;
    std::string    get_family_name() const;
    size_type      size  (reference_element hat_K) const;
    const_iterator begin (reference_element hat_K) const;
    const_iterator end   (reference_element hat_K) const;
    friend std::ostream& operator<< (std::ostream&, const quadrature&);
protected:
    quadrature_option_type     _options;
    mutable quadrature_on_geo  _quad        [reference_element::max_size];
    mutable std::vector<bool>  _initialized;
    void _initialize (reference_element hat_K) const;
private:
    quadrature (const quadrature&);
    quadrature operator= (const quadrature&);
};
```

8.7 numbering - global degree of freedom numbering

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

Synopsys

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

```
class numbering : public smart_pointer<numbering_rep> {
public:

// typedefs:
    typedef numbering_rep                rep;
    typedef smart_pointer<numbering_rep> base;
    typedef size_t                       size_type;

// allocators:

    numbering (std::string name = "");
    numbering (numbering_rep* ptr);

    virtual ~numbering() {}

// accessors:

    std::string name() const;
    virtual
    size_type ndof (
              size_type  mesh_map_dimension,
        const size_type* mesh_n_geo,
        const size_type* mesh_n_element) const;

    virtual
    size_type idof (
        const size_type*      mesh_n_geo,
        const size_type*      mesh_n_element,
        const geo_element&    K,
        size_type             i_dof_local) const;

    virtual
    void idof (
        const size_type*        mesh_n_geo,
```

```
            const size_type*        mesh_n_element,
            const geo_element&      K,
            std::vector<size_type>& i_dof) const;

        virtual bool is_continuous() const;
        virtual bool is_discontinuous() const { return !is_continuous(); }

    // i/o:

        void dump(std::ostream& out = std::cerr) const;
    };
```

8.8 `reference_element` - reference element

(Source file: '`nfem/basis/reference_element.h`')

Synopsys

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 enum_type { p, t, q ..., H, ...};
        // in an automatically generated file :

        typedef enum {
            p = 0,
            e = 1,
            t = 2,
            q = 3,
            T = 4,
            P = 5,
```

```
        H = 6,
        max_size = 7
    } enum_type;

    typedef enum {
        Lagrange = 0,
        Hermite = 1,
        dof_family_max =2
    } dof_family_type;

// constants:

    static const size_type not_set;
    static const size_type max_n_subgeo      = 12;
    static const size_type max_subgeo_vertex = 8;

// allocators/deallocators:

    reference_element (enum_type x = max_size);

// accessors:

    enum_type type() const;
    char name() const;
    size_type dimension() const;
    friend Float measure (reference_element hat_K);
    size_type size() const;
    size_type n_edge() const;
    size_type n_face() const;
    size_type n_subgeo(size_type subgeo_dim) const;
    size_type subgeo_size(size_type subgeo_dim, size_type i_subgeo) const;
    size_type subgeo_local_vertex(size_type subgeo_dim, size_type i_subgeo, size_ty
    size_type heap_size() const;
    size_type heap_offset(size_type subgeo_dim) const;

    void set_type (enum_type x);
    void set_type (size_type n_vertex, size_type dim);
    void set_name (char name);

// data:
protected:

    enum_type _x;

// constants:

    // these constants are initialized in
    //   "reference_element_declare.c"
    // automatically generated.
```

```
    static const char _name                     [max_size];
    static const size_type _dimension           [max_size];
    static const size_type _n_subgeo            [max_size][4];
    static const size_type _subgeo_size         [max_size*4*max_n_subgeo];
    static const size_type _subgeo_local_vertex [max_size*4*max_n_subgeo*max_subgeo
    static const size_type _heap_size           [max_size];
    static const size_type _heap_offset         [max_size][4];
    friend std::istream& operator>>(std::istream&, class geo_element&);
};
```

8.9 geo_element - element of a mesh

(Source file: 'nfem/basis/geo_element_v1.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 Section 8.8 [reference_element internal], page 123). 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 Section 5.2 [geo class], page 42).

Example

This is the test of geo_element:

```
geo_element 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;
}
```

Implementation

```
class geo_element : public reference_element {
public:

// allocators/deallocators:

    geo_element(enum_type t = max_size);
    geo_element(const geo_element&);
    explicit geo_element (const class tiny_element&);
    void copy (const geo_element&);
    void copy (const class tiny_element&);
    geo_element& operator = (const geo_element&);
    ~geo_element();

// accessors:

    size_type index() const;
    size_type operator [] (size_type i) const;
    size_type side(size_type i_side) const;
    void build_side(size_type i_side, geo_element& S) const;


    size_type edge (size_type i_edge) const;
    size_type face (size_type i_face) const;

    size_type subgeo(const geo_element& S) const;
    size_type subgeo (size_type subgeo_dim, size_type i_subgeo) const;
    size_type subgeo_vertex (size_type subgeo_dim, size_type i_subgeo,
                             size_type i_subgeo_vertex) const;
    void build_subgeo(size_type subgeo_dim, size_type i_subgeo, geo_element& S) con
    size_type subgeo_local_index(const geo_element& S) const;

// modifiers:

    void set_type (enum_type t);
    void set_type (size_type sz, size_type dim);
    void set_name (char      name);

    void set_index(size_type idx);
    size_type& operator [] (size_type i);
    void set_side (size_type i_side, size_type idx);

    void set_edge (size_type i_edge, size_type idx);
    void set_face (size_type i_face, size_type idx);
    void set_subgeo (size_type subgeo_dim, size_type i_subgeo,
                             size_type idx);
    void set_subgeo(const geo_element& S, size_type idx);
```

```
// inputs/outputs:

    friend std::istream& operator >> (std::istream&, geo_element&);
    friend std::ostream& operator << (std::ostream&, const geo_element&);
    std::ostream& dump(std::ostream& s = std::cerr) const;
    void check() const;

// data:
protected:
    size_type *_heap;

// memory management:
    void _heap_init();
    void _heap_close();
};
```

8.10 point - Point reference element

(Source file: 'nfem/basis/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 size_t measure = 1;
```

8.11 edge - Edge reference element

(Source file: 'nfem/basis/edge.icc')

Description

The edge reference element is K = [0,1].

```
0---------1   x
```

Implementation

```
const size_t dimension = 1;
const size_t measure = 1;
const size_t n_vertex = 2;
const Float vertex [n_vertex][dimension] = {
        { 0 },
        { 1 } };
```

8.12 `triangle` - Triangle reference element

(Source file: '`nfem/basis/triangle.icc`')

Description

The triangle reference element is

```
K = { 0 < x < 1 and 0 < y < 1-x }
```

Numbering

```
y

2
| +
|   +
|     +
|       +
0---------1   x
```

Implementation

```
const size_t dimension = 2;
const Float  measure = 0.5;
const size_t n_vertex = 3;
const Float vertex [n_vertex][dimension] = {
        { 0, 0 },
        { 1, 0 },
        { 0, 1 } };
const size_t  n_edge = 3;
const size_t edge [n_edge][2] = {
        { 0, 1 },
        { 1, 2 },
        { 2, 0 } };
```

8.13 `quadrangle` - Quadrangular reference element

(Source file: '`nfem/basis/quadrangle.icc`')

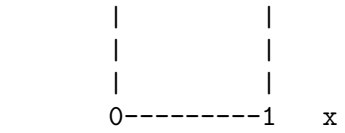
Description

The quadrangular reference element is [0,1]^2.

Numbering

```
y

3---------2
|         |
```



Implementation

```
const size_t dimension = 2;
const Float  measure = 4;
const size_t n_vertex = 4;
const Float vertex [n_vertex][dimension] = {
        { -1,-1 },
        {  1,-1 },
        {  1, 1 },
        { -1, 1 } };
const size_t  n_edge = 4;
const size_t edge [n_edge][2] = {
        { 0, 1 },
        { 1, 2 },
        { 2, 3 },
        { 3, 0 } };
```

8.14 `tetra` - Tetraedra reference element

(Source file: '`nfem/basis/tetra.icc`')

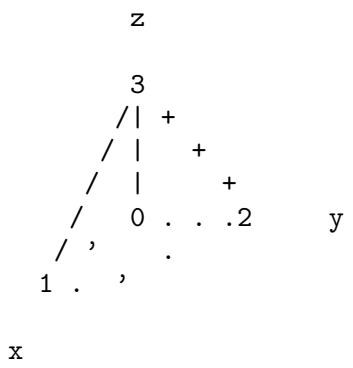
Description

The tetraedra reference element is

```
K = { 0 < x < 1 and 0 < y < 1-x and 0 < z < 1-x-y }
```

Numbering

The orientation is such that triedra (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 = Float(1.)/Float(6.);
const size_t n_vertex = 4;
const Float vertex [n_vertex][dimension] = {
        { 0, 0, 0 },
        { 1, 0, 0 },
        { 0, 1, 0 },
        { 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 } };
```

8.15 `prism` - Prism reference element

(Source file: '`nfem/basis/prism.icc`')

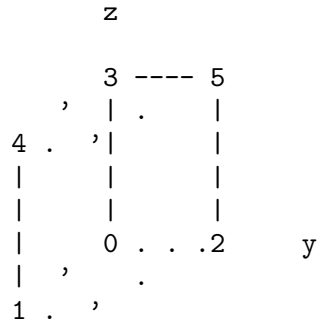
Description

The prism reference element is

```
K = { 0 < x < 1 and 0 < y < 1-x and -1 < z < 1 }
```

Numbering

The orientation is such that triedra (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.



```
    x
```

Implementation

```
const size_t dimension = 3;
const Float  measure = 1;
const size_t n_vertex = 6;
const Float vertex [n_vertex][dimension] = {
        { 0, 0,-1 },
        { 1, 0,-1 },
        { 0, 1,-1 },
        { 0, 0, 1 },
        { 1, 0, 1 },
        { 0, 1, 1 } };
const size_t  n_face = 5;
const size_t face [n_face][4] = {
        { 0, 2, 1, size_t(-1) },
        { 0, 3, 5, 2 },
        { 0, 1, 4, 3 },
        { 3, 4, 5, size_t(-1) },
        { 1, 2, 5, 4 } };
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 } };
```

8.16 `hexa` - Hexaedra reference element

(Source file: '`nfem/basis/hexa.icc`')

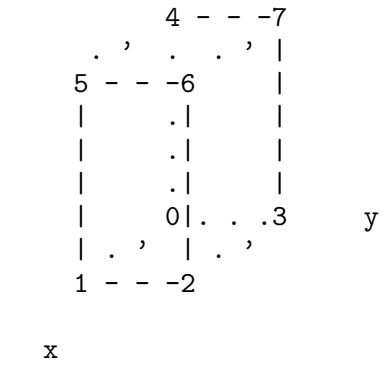
Description

The hexa reference element is [-1,1]^3.

Numbering

The orientation is such that triedra (01, 03, 04) 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.

```
                z
```



Implementation

```
const size_t dimension = 3;
const Float  measure = 8;
const size_t n_vertex = 8;
const Float vertex [n_vertex][dimension] = {
        {-1,-1,-1 },
        { 1,-1,-1 },
        { 1, 1,-1 },
        {-1, 1,-1 },
        {-1,-1, 1 },
        { 1,-1, 1 },
        { 1, 1, 1 },
        {-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 } };
```

8.17 occurence, smart_pointer - memory management

(Source file: 'util/lib/smart_pointer.h')

Description

Here is a convenient way to implement a true copy semantc, 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.

Implementation

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

// allocators:

        smart_pointer (T* p = 0);
        smart_pointer (const smart_pointer&);
        ~smart_pointer ();
        smart_pointer& operator= (const smart_pointer&);

// accessors:

        const T* pointer    () const;
        const T& data       () const;
        const T* operator-> () const;
        const T& operator*  () const;

// modifiers:

        T* pointer ();
        T* operator-> ();
        T& data ();
        T& operator* ();

// implementation:

private:
        struct counter {
                T*  _p;
                int _n;
                counter (T* p = 0);
                ~counter ();
                int operator++ ();
                int operator-- ();
```

```
        };
        counter *_count;
};
```

8.18 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 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;
    };

// assignement:

    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);
    }
    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;
};
```

8.19 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 emmited. 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 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))
    {
        warning_macro ("stack_allocator cstor");
    }
    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()
    {
```

```
        warning_macro ("stack_allocator dstor");
        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;
    };

// assignement:

    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);
    }
    void destroy (pointer p)
    {
        // call destructor directly:
        (p)->~T();
    }

// allocate raw memory

    pointer allocate (size_type n, const void* = NULL)
    {
        warning_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) {
            warning_macro ("stack is full: throwing...");
```

```
            throw std::bad_alloc();
        }
        return pointer (p);
    }
    void deallocate (pointer p, size_type n)
    {
        warning_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)
        {
          warning_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)
        {
          warning_macro ("stack_allocator::mem_info cstori: size="<<max_size);
        }
        ~handler_type()
        {
          warning_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();
}
```

8.20 typename_macro, pretty_typename_macro - type demangler and pretty printer

(Source file: 'util/lib/pretty_name.h')

Description

These preprocessor macro-definitions are usefull 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::tring 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.

Example

```
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);

/// @brief get string from a type, with an optional pretty-printing for complex typ
#define        typename_macro(T) typeid_name(typeid(T).name(), false)
#define pretty_typename_macro(T) 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_ma
template <class T> std::string pretty_typename_of (T x) { return pretty_typename_ma
```

8.21 acinclude – autoconf macros

(Source file: 'config/acinclude.m4')

Description

These macros test for particular system featutres 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 *subsitutions*. See `autoconf` documentation for how to use such variables.

Synopsis

Follows a list of particular check required for a successfull 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 substitues INCLUDES_GINAC and LADD_GINAC for adding in CFLAGS and LDFLAGS, 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 substitues INCLUDES_CLN and LADD_CLN for adding in CFLAGS and LDFLAGS, 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 substitues INCLUDES_TAUCS and LADD_TAUCS for adding in CXXFLAGS and LDFLAGS, respectively, If not, set the shell variable to "no". Includes and libraries options are given shell variable $rheo_ldadd_taucs and $rheo_incdir_taucs. These shell variables could be set for instance by appropriates "–with-taucs-ldadd="'rheo_ldadd_taucs' and "–with-taucs-includes="'rheo_incdir_taucs' options.

`RHEO_CHECK_BOOST`

Check to see if boost headers exists. If so, set the shell variable "rheo_have_boost" to "yes", defines HAVE_BOOST and substitues 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_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 substitues INCLUDES_ZLIB and LADD_ZLIB for adding in CXXFLAGS and LDFLAGS, 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 substitues INCLUDES_SPOOLES and LADD_SPOOLES for adding in CXXFLAGS and LDFLAGS, 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_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 substitues INCLUDES_UMFPACK and LADD_UMFPACK for adding in CXXFLAGS and LDFLAGS, 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 `-I`*dir_malloc_dbg*`/include` to CFLAGS, add *dir_malloc_dbg*`/lib/libmalloc_dbg.a` to LDFLAGS. 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 substitues 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 wheter "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 fonctionality is usefull to open a pipe stream class, as "pstream(3)".

`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 funtions

```
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 substitues FLEX to "flex" and FLEXLEXER_H to the full path for FlexLexer.h If not, set the shell variable no "no".

RHEO_PROG_CC_KAI

Check wheter we are using KAI C++ compiler. If so, set the shell variable "ac_cv_prog_kcc" to "yes". 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_CRAY

Check wheter we are using CRAY C++ compiler. If so, set the shell variable "ac_cv_prog_cray_cc" to "yes" and defines HAVE_CRAY_CXX. If not, set the shell variable no "no". The shell variable is also exported for sub-shells.

RHEO_PROG_CC_DEC

Check wheter we are using DEC C++ compiler. If so, set the shell variable "ac_cv_prog_dec_cc" to "yes". If not, set the shell variable no "no". The shell variable is also exported for sub-shells, such as ltconfig from libtool.

RHEO_RECOGNIZE_CXX

Check wheter we are able to recognize the C++ compiler. Tested compilers:

```
The KAI  C++ compiler that defines: __KCC     (KCC-3.2b)
The GNU  C++ compiler that defines: __GNUC__  (egcs-1.1.1)
The CRAY C++ compiler that defines: cray
The DEC  C++ compiler that defines: __DECCXX
```

If so, substitue RECOGNIZED_CXX to a specific compiler's rule file, e.g, "${top_srcdir}/config/gnu_cxx.mk for a subsequent Makefile include. If not, substitue 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 requiere the cc compiler, such as "alloca.c" use some specific makefile rule.

usage example:

```
AC_PROG_CC(gcc cc cl)
AC_PROG_CXX(c++ g++ cxx KCC CC CC cc++ xlC aCC)
RHEO_RECOGNIZE_CXX
```

RHEO_OPTIMIZE_CXX

Set some optimization flags associated to the recognized C++ compiler and platform.

RHEO_CHECK_LATEX_HYPEREF

Check whether the hyperref LaTeX package exists and set the corresponding shell value "rheo_have_latex_hyperref" and HAVE_LATEX_HYPEREF (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", defines HAVE_MPI and substitues MPIRUN to "mpirun" and RUN to "mpirun -np 2", INCLUDES_MPI and LDADD_MPI. If not, set the shell variable no "no".

`RHEO_CHECK_PARMETIS`

Check for the "parmetis" and "metis" libraries. Defines HAVE_PARMETIS and substitues INCLUDES_MPI and LDADD_MPI. Requires the MPI library.

`RHEO_CHECK_SCOTCH`

Check for the "scotch" distributed mesh partitionner libraries. Defines HAVE_SCOTCH and substitues INCLUDES_SCOTCH and LDADD_SCOTCH. Requires the MPI library.

`RHEO_CHECK_BLAS`

Check for the "blas" basic linear algebra subroutines library. Defines HAVE_BLAS and substitues LDADD_BLAS and INCLUDES_BLAS.

`RHEO_CHECK_PASTIX`

Check for the "pastix" distributed direct solver libraries. Defines HAVE_PASTIX and substitues INCLUDES_PASTIX and LDADD_PASTIX. Requires the MPI and SCOTCH libraries.

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`.

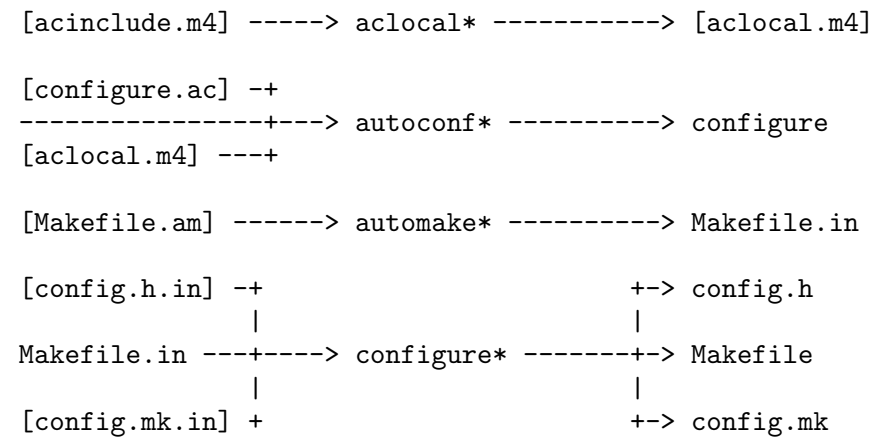
9.1 How to regenerate the `configure` script

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:



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
```

Its mean 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 rnon-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
```

Its mean 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 checkout 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 checkout Makefile.am
```

You restaure the last available version of the missing file in the most up-to-date Rheolef distribution.

Appendix A GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) 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 this service 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 make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. 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.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

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

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute 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 and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions

for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.
4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you

indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), 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 distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the 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 a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, 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.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE 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.

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 convey 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 an idea of what it does.*
> Copyright (C) 19*yy* *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 2
> 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, write to the Free Software
> Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) 19*yy* *name of author*
> Gnomovision 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, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright
> interest in the program 'Gnomovision'
> (which makes passes at compilers) written
> by James Hacker.
>
> *signature of Ty Coon*, 1 April 1989
> Ty Coon, President of Vice

This 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 Library General Public License instead of this License.

Concept Index

Program Index

Class Index

Form Index

2

C

D

G

I

M

Approximation Index

B

H

P

Function Index

File Format Index

Related Tool Index

P

Q

S

T

U

V

X

Short Contents

Table of Contents