Rheolef, the finite element system.

Reference manual
Version 6.1
15 January 2012

by Pierre Saramito

Copyright © 1996, 1998, 2001, 2002 by Pierre Saramito. All rights reserved.

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

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

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.

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

The reference manual is generated in html format:

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

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

```
man field
man 2rheolef 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="$HOME/sys/bin:$PATH"
export LD_LIBRARY_PATH="$HOME/sys/lib:$LD_LIBRARY_PATH"
export MANPATH="$HOME/sys/man:$MANPATH"
```

assuming you are using the sh or the bash unix interpreter. Conversely, if you are using csh or tcsh, add at the bottom of your '.cshrc' file:

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

If you are 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 Requiered libraries

All the required libraries have precompiled packages under Debian GNU/Linux and Ubuntu systems. If you are running a system where these libraries are not available, please consult the corresponding home pages and install instructions.

a) Rheolef widely uses the `boost` library, that provides usefull extensions of the c++ standard template library. `boost` at `http://www.boost.org` .

b) The use of implicit numerical schemes leads to the resolution of large sparse linear systems. Rheolef installation optionally requieres `mumps` as direct solver library. `mumps` is available at `http://graal.ens-lyon.fr/MUMPS` .

c) Rheolef implements the characteristic method and interpolation from one mesh to another: these features leads to sophisticated algorithms and octree-type data structures. The `cgal` library provide easy access to efficient and reliable geometric algorithms in the form of a C++ library. `cgal` is available at `http://www.cgal.org` .

2.4 Highly recommended libraries

All the highly recommended libraries have precompiled packages under Debian GNU/Linux and Ubuntu systems. With all these libraries, Rheolef with run optimally. Otherwise, it will build and run also, but with some loss of performances.

a) Rheolef bases on `mpi`, the message passing interface standard. When no `mpi` libraries are available, the distributed features are turned off.

b) Rheolef bases on `scotch` for distributed mesh partitioning. `scotch` is available at `http://www.labri.fr/perso/pelegrin/scotch` . When the `scotch` library is not available, the distributed features are turned off.

c) Rheolef installation optionally uses `trilinos/ifpack` as a library of preconditioner used for iterative solvers. `trilinos` is available at `http://trilinos.sandia.gov` . When `trilinos` is not available, a less efficient builtin IC0 preconditioner is used.

2.5 Run-time optional tools

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

```
gmsh
gnuplot
mayavi
paraview
```

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

2.6 Build-time extra tools for rheolef developpers

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

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

2.7 Portability issues

Rheolef is based on STL and its extensions, the BOOST library. Therefore you need a compiler that supports the ANSI C++ standards. STL and Rheolef especially depend heavily on the template support of the compiler. Most recent c++ 2012 standard features are recommanded but still optional: when available, it allows an elegant matrix concatenation syntax via the `std::initializer_list` class.

Current distributed version of Rheolef has been heavily tested with GNU C++ and INTEL C++ on debian, ubuntu and suse GNU/Linux systems for various 32 and 64 bits architectures, up to 100 distributed memories and processes.

The GNU C++ compiler is a free software available at `http://www.gnu.org`. The Intel C++ compiler is a privative software.

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

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

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

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

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

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

```
CXX=icpc CXXFLAGS=-I/usr/local/include LIBS="-L/usr/local/lib -lz" ./configure
```

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

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.

`--enable-optim`
: Turns compile-time optimization to maximum available. Include architecture-dependent compilation (e.g. on intel, executable are less portable accross intel variants). Default is on.

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

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

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

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

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

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

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

`--with-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.

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

2.8 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.

rpath issue for debian package: seems to be buggy with DESTDIR!=""

2.9 The directory structure

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

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 `branch` – handle a family of fields

(Source file: '`nfem/pbin/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 5.6 [geo class], page 46 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

`-umin` *float*
`-umax` *float*
: set the solution range for the `gnuplot` driver. By default this range is computed from the first field of the branch, and this could be problematic when this field is initialy zero.

`-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
  ...
```

4.2 `field` – plot a field

(Source file: '`nfem/pbin/field.cc`')

Synopsis

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

Description

Read and output a finite element field from file.

Example

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

```
field box.field
```

Input file specification

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

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

`-I`*dir*

`-I` *dir* Add *dir* to the rheolef file search path. This option is usefull e.g. when the mesh .geo and the .field fikes are in different directories. This mechanism initializes a search path given by the environment variable '`RHEOPATH`'. If the environment variable '`RHEOPATH`' is not set, the default value is the current directory.

`-catchmark` *label*
Jump accross the file to the specifield label. Label start at the begining of a line, preceded by a '`#`' mark (see see Section 7.14 [catchmark algorithm], page 121).

Output file format options

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

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

`-gmsh-pos`
output field on standard output stream in gmsh-pos text file format, suitable for mesh adaptation purpose.

`-bamg-bb` output field on standard output stream in bamg-bb text file format, suitable for mesh adaptation purpose.

`-image-format string`
The argument is any valid image format, such as bitmap `png`, `jpg`, `gif` or vectorial `ps` or `pdf` image file formats, and that could be handled by the corresponding render. The output file is e.g. '*basename*`.png`' when *basename* is the name of the mesh, or can be set with the `-name` option.

Getting information

`-min`

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

Render options

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

`-mayavi` use `mayavi` tool. This is the default for tridimensional geometries.

Rendering options

`-color`
`-gray`
`-black-and-white`
`-bw` Use (color/gray scale/black and white) rendering. Color rendering is the default.
`-elevation`
`-noelevation`
For two dimensional field, represent values as elevation in the third dimension. The default is no evelation.

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

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

`-fill` isoline intervals are filled with color. This is the default.
`-nofill` draw isolines by using lines.
`-cut`
`-nocut` Cut by a specified plane. The cutting plane is specified by its origin point and normal vector. This option requires the `mayavi` code.
`-origin` *float* [*float* [*float*]]
set the origin of the cutting plane. Default is (0.5, 0.5, 0.5).

`-normal` *float* [*float* [*float*]]
set the normal of the cutting plane. Default is (1, 0, 0).

`-iso` [*float*]
do draw 3d isosurface. When the optional float is not provided, a median value is used. This option requires the `mayavi` code.

`-noiso` do not draw isosurface.
`-n-iso` *int*
For 2D visualizations, the isovalue table contains regularly spaced values from fmin to fmax, the bounds of the field.

`-proj` Convert all selected fields to Pk-continuous approximation by using a L2 projection.
`-round` [*float*]
Round the input up to the specifiexd precision. This option, combined with `-field`, leads to a round filter. Usefull for non-regression test purpose, in order to compare numerical results between files with a limited precision, since the full double precision is machine-dependent.

`-n-iso-negative` *int*
The isovalue table is 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.

`-subdivide int`
When using a high order geometry, the number of points per edge used to draw a curved element. Default value is the mesh order.

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

`-velocity`
Render vector-valued fields as arrows using `mayavi`.

Component extraction

`-comp` *int* Extract a component of a vector-valued field.

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 4.3 [geo command], page 17) 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.

Input file format options

TODO

File format conversions

TODO

Getting information

TODO

4.3 `geo` - plot a finite element mesh

(Source file: '`nfem/pbin/geo.cc`')

Synopsis

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

Description

Plot or upgrade a finite element mesh.

Examples

Plot a mesh:

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

Plot a mesh into a file:

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

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

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

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

Input file specification

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

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

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

`-I`*dir*

`-I` *dir* Add *dir* to the rheolef file search path. This mechanism initializes a search path given by the environment variable '`RHEOPATH`'. If the environment variable '`RHEOPATH`' is not set, the default value is the current directory.

`-check` Check that element orientation is positive.

Input format options

`-if` *format*

`-input-format` *format*

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

Render specification

`-gnuplot` use gnuplot tool. This is the default.

`-mayavi` use `mayavi` tool.

Render options

`-lattice`
`-nolattice`
: When using a high order geometry, the lattice inside any element appears. Default is on;

`-subdivide int`
: When using a high order geometry, the number of points per edge used to draw a curved element. Default value is the mesh order.

`-full`
`-nofull` All internal edges appears, for 3d meshes. Default is off;
`-fill`
`-nofill` Fill mesh faces using light effects, when available.
`-stereo`
`-[no]stereo`
: Rendering mode suitable for red-blue anaglyph 3D stereoscopic glasses. Option only available with `mayavi`.

`-cut`
`-nocut` cut by plane and clip (with `mayavi` only).

Output file format options

`-geo` output mesh on standard output stream in geo text file format, instead of plotting it.
`-upgrade` Convert from a old geo file format to the new one.
`-gmsh` output mesh on standard output stream in gmsh text file format, instead of plotting it.
`-image-format string`
: The argument is any valid image format, such as bitmap `png`, `jpg`, `gif` or vectorial `ps` or `pdf` image file formats, and that could be handled by the corresponding render. The output file is e.g. '*basename*`.png`' when *basename* is the name of the mesh, or can be set with the `-name` option.

Others options

`-add-boundary`
: check for a domain named "boundary"; If this domain does not exists, extract the boundary of the geometry and append it to the domain list. This command

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

`-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.

Inquire options

`-min-element-measure`
`-max-element-measure`
print the smallest (resp. largest) element measure and then exit.

Geo file format

TODO

4.4 `msh2geo` - convert gmsh mesh in geo format

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

Synopsis

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

Description

Convert a gmsh '`.msh`' into '`.geo`' one. The output goes to standart output. See the `gmsh` documentation for a detailed description of the '`.mshcad`' input file for `gmsh`.

Examples

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

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

Coordinate system option

Most of rheolef codes are coordinate-system independant. The coordinate system is specified in the geometry file '`.geo`'.

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

Notes

Pk triangle, when k>=5, may have internal nodes renumbered: from the

Gmsh documentation

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

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

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

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

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

Todo

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

See also hexa edges orient and faces numbers and orient.

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

Support for high order >= 6 element ? not documented in gmsh, but gmsh supports it at run

4.5 `bamg2geo` - convert bamg mesh in geo format

(Source file: '`nfem/sbin/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 4.3 [geo command], page 17. With the `-noupgrade`, a version 1 file format is assumed.

`-dom` *dom1* `...` `-dom` *domN*

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

(Source file: '`nfem/sbin/mkgeo_ball`')

Synopsis

```
mkgeo_ball options [n]
```

Example

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

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

or in one comand line:

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

Description

This command is usefull when testing programs on simple geometries. Invocation is similar to `mkgeo_grid` (see Section 4.7 [mkgeo_grid command], page 24, Section 4.8 [mkgeo_ugrid command], page 27). It calls `gmsh` as unstructured mesh generator. It avoid the preparation of an input file for a mesh generator. The optional n argument is an integer that specifies the subdivision in each direction. By default n=10. The mesh files goes on standard output.

Element type options

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

The mesh order

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

The geometry

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

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

Boundary domains

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

Others options

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

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

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

Synopsis

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

Example

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

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

or in one 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

`-sides`
`-nosides` The boundary sides are representd by domains: `left`, `right`, `top`, `bottom`,`front` and `back`.
`-boundary`
`-noboundary`
This option defines a domain named `boundary` that groups all sides.

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

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

Regions

`-region`
`-noregion`
The whole domain is 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`
`-nocorner`
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`'.

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

File format option

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

(Source file: '`nfem/sbin/mkgeo_ugrid`')

Synopsis

```
mkgeo_ugrid options [n]
```

Example

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

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

or in one comand line:

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

Description

This command is usefull when testing programs on simple geometries. Invocation is similar to `mkgeo_grid` (see Section 4.7 [mkgeo_grid command], page 24). It calls `gmsh` as unstructured mesh generator. It avoid the preparation of an input file for a mesh generator. The optional *n* argument is an integer that specifies the subdivision in each direction. By default *n*=10. The mesh files goes on standard output.

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

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

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

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

Element type options

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

The geometry

The geometry can be any [a,b] segment, [a,b]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_ugrid -t 10 -a -2 -b 2 -c -1.5 -d 1.5 | geo -
```

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

Boundary domains

`-sides`
`-nosides` The boundary sides are representd by domains: `left`, `right`, `top`, `bottom`,`front` and `back`.
`-boundary`
`-noboundary`
This option defines a domain named `boundary` that groups all sides.

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

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

Regions

`-region`
`-noregion`
The whole domain is 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_ugrid -t 10 -region | geo -
```

Corners

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

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

The mesh order

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

Others options

`-clean` clear temporary files (this is the default).
`-noclean` does not clear temporary files.

4.9 `bamg` - bidimensional anisotropic mesh generator

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

Synopsis

`bamg` *options* `-g` *input*`[.bamgcad] -o` *ouput*`[.bamg]`

Example

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

Outline of the commands

This software can

`create` a mesh from a geometry
`adapt` a mesh from a background mesh using a metric or solution file
`do smoothing`
: to make quadrilaterals and to split internal edge with 2 boundary vertices in an existing mesh (in this case the metric is use to change the definition of the element's quality).

`construct`
: just a metric file, if you have an other mesher

`do the P1 interpolation`
: of the solution on another mesh:

Create a mesh from a geometry

`input` file with `-g filename` (file type `DB mesh`).
`output` file arguments `-oxxxx filename` where `xxxx` is the type of output file (see below for more details).

`smoothing, quad, utility and other parameters.`

In addition, metric specication may be prescribed with the help of `-M filename` argument (file type `Metric`). All the options are described below.

Adapt a mesh from a background mesh using a metric or solution file

`input` files `-b filename` where the suffixe of the file define the type of the file '`.amdba`', '`.am_fmt`', '`.am`', '`.ftq`', '`.nopo`' and otherwise the file is a `BD mesh` file.
`input` '`-M filename`' (file type `Metric`) or `-MBB filename` or `-Mbb filename` (solution file type `BB` or `bb`).
`metric` control parameters, `-err val`, `-errg val`,....
`output` file arguments reqired
`interpolation, smoothing, quad,`
utility parameter, and other parameter

do smoothing, to make quadrilaterals: and to split internal edge with 2 boundary vertices in an existing mesh (in this case the metric is use to change the definition of the element's quality).

`input` files `-r filename`(file type `DB mesh`).
`-M filename`
(file type `Metric`).
`output` file `-o filename` (file type `DB mesh`).
`some other parameter:`
`-thetaquad val:`
to create quad with 2 triangles
`-2:` to create the submesh with mesh size h = h/2
`-2q:` to split all triangles in 3 quad. and to split all quad. in 4 quad.
`-NbSmooth ival:`
to change the number of smoothing iteration (3 by default if the metric eld is set with arguments : `-M` or `-Mbb`, 0 otherwise.
`...`

construct just a metric file, if you have an other mesher:

`input` files `-r filename` (file type `DB mesh`).
`--Mbb filename`
or `-MBB filename`
`+ all the arguments`
of the metric construction
`output` file `-oM filename` (file type `Metric`).

do the P1 interpolation of the solution on another mesh:

`input` files `-r filename` (file type `DB mesh`).
`--rBB filename`
or `-rbb filename`
`output` file `-wBB filename` or `-wbb filename`

Metric computatiob options

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

`-RelError`
: compute the metric with a relative error. This is the default.

`-CutOff float`
: the cut-off value used for the relative error criteria. Default value is 1e-5.

`-AbsError`
: compute the metric with a relative error

4.10 `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.

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

`--have-old-code`

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

5 Classes

5.1 band - compute the band arround a level set

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

Description

Given a function fh defined in a domain Lambda, compute the band of elements intersecting the level set defined by {x in Lambda, fh(x) = 0}. This class is used for solving problems defined on a surface described by a level set function (See Section 7.5 [level_set algorithm], page 107).

Accessors

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

Band topology and domains

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

Implementation

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

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

// allocators:

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

/// accessors:

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

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

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

5.2 branch - a parameter-dependent sequence of field

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

Description

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

Implementation

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

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

// allocators:

  branch_basic ();
```

```
  branch_basic (const std::string& parameter_name, const std::string& u0);
  branch_basic (const std::string& parameter_name, const std::string& u0, const std
  ~branch_basic();

// accessors:

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

// modifiers:

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

// input/output:

  // get/set current value
  __obranch<T,M> operator()  (const T& t, const field_basic<T,M>& u0);
  __obranch<T,M> operator()  (const T& t, const field_basic<T,M>& u0, const field_b

  __iobranch<T,M> operator() (T& t, field_basic<T,M>& u0);
  __iobranch<T,M> operator() (T& t, field_basic<T,M>& u0, field_basic<T,M>& u1);

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

5.3 `characteristic` - the Lagrange-Galerkin method implemented

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

Synopsys

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

Example

The following code compute the Riesz representer (see Section 7.7 [riesz algorithm], page 109), denoted by `lh` of u(x)=uh(x+dh(x)) where `ah` is the deformation vector field. The deformation field `dh=-dt*uh` in Lagrange-Galerkin methods, where `ah` is the advection field and `dt` a time step.

```
geo omega;
field dh = ...;
field uh = ...;
```

```
  characteristic X (dh);
  field lh = riesz(Xh, compose(uh, X));
```

The Riesz representer is the `lh` vector field defined by:

```
        lh(i) = integrate uh(x+dh(x)) phi_i(x) dx
```

where phi_i is the i-th basis function in the space Xh 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 Xh (order 1: trapeze, order 2: simpson, etc). Recall that this choice of quadrature formulae guaranties inconditional stability at any polynomial order. Alternative quadrature formulae or order can be used by using the additional quadrature option argument to the `riesz` function.

Implementation

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

// allocator:

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

// accesors:

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

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

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

Implementation

```
template<class T, class M>
inline
field_o_characteristic<T,M>
compose (const field_basic<T,M>& uh, characteristic_basic<T,M>& X)
```

Implementation

```
template <class T, class M>
```

```
field_basic<T,M>
riesz (
    const space_basic<T,M>&             Xh,
    const field_o_characteristic<T,M>&  f,
    quadrature_option_type              qopt
       = quadrature_option_type(quadrature_option_type::max_family,0));
```

5.4 `field` - piecewise polynomial finite element field

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

Description

Store degrees of freedom associated to a mesh and a piecewise polynomial approximation, with respect to the numbering defined by the underlying Section 5.7 [space class], page 53.

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

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

Interpolation

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

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

Linear algebra

Linear algebra, such as `uh+vh`, `uh-vh`, `lambda*uh + mu*vh`, ...are supported. The Euclidian dot product writes simply `dot(uh,vh)`. The application of a bilinear form (see Section 5.5 [form class], page 43) writes `m(uh,vh)` and is equivalent to `dot(m*uh,vh)`.

Non-linear algebra

Non-linear operations, such as `sqrt(uh)` or `1/uh` are also available. Notice that non-linear operations do not returns in general picewise polynomials: the value returned by `sqrt(uh)` is the Lagrange interpolant of the square root of `uh`. These operators applies directly to the set of degrees-of-freedom (dofs, see below). All standard unary and binary math functions `abs, cos, sin`... are available on fields. Also `sqr(uh)`, the square of a field, and `min(uh,vh)`, `max(uh,vh)` are provided. Binary functions can be used also with a scalar, as in

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

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

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

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

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

Access by domain

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

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

Extraction of the trace as a field is also possible:

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

The space associated to the trace writes `wh.get_space()` and is equivalent to `space(omega["boundary"], "P1")`. See see Section 5.7 [space class], page 53.

Vector valued field

A vector-valued field contains several components, as:

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

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

Tensor valued field

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

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

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

General multi-component interface

A general multi-component field writes:

```
space Th (omega, "P1d", "tensor");
space Vh (omega, "P2", "vector");
space Qh (omega, "P1");
space Xh = Th*Vh*Qh;
field xh (Xh);
field tau_h = xh[0]; // tensor-valued
field uh    = xh[1]; // vector-valued
field qh    = xh[2]; // scalar
```

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

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

Blocked and unblocked arrays

The field class contains two arrays of degrees-of-freedom (dof) associated respectively to blocked and unknown dofs. Blocked dofs corresponds to Dirichlet boundary conditions, as specified by space (See see Section 5.7 [space class], page 53). For simpliity, direct public access to these array is allowed, as `uh.b` and `uh.u`: see see Section 5.19 [vec class], page 86.

Low-level degree-of-freedom access

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

For performances, a STL-like iterator interface is available, with `uh.begin_dof()` and `uh.end_dof()` returns iterators to the arrays of dofs on the current processor. See see Section 5.11 [array class], page 66 for more about distributed arrays.

File format

TODO

Implementation note

The field expression use the expression template technics in order to avoid temporaries when evaluating complex expressions.

Implementation

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

    typedef typename std::size_t            size_type;
    typedef T                               value_type;
    typedef typename scalar_traits<T>::type float_type;
    typedef geo_basic  <float_type,M>       geo_type;
    typedef space_basic<float_type,M>       space_type;
    typedef space_constant::valued_type     valued_type;
    class iterator;
    class const_iterator;

// allocator/deallocator:

    field_basic();

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

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

    field_basic                   (const field_indirect<T,M>&);
    field_basic<T, M>& operator=  (const field_indirect<T,M>&);
    field_basic                   (const field_indirect_const<T,M>&);
    field_basic<T, M>& operator=  (const field_indirect_const<T,M>&);
    field_basic                   (const field_component<T,M>&);
    field_basic<T, M>& operator=  (const field_component<T,M>&);
    field_basic                   (const field_component_const<T,M>&);
    field_basic<T, M>& operator=  (const field_component_const<T,M>&);
    template <class Expr> field_basic                   (const field_expr<Expr>&);
    template <class Expr> field_basic<T, M>& operator=  (const field_expr<Expr>&);
    field_basic<T, M>& operator=  (const T&);

// initializer list (c++ 2011):

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

// accessors:
```

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

// accessors & modifiers to unknown & blocked parts:

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

// accessors to extremas:

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

// accessors by domains:

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

// accessors by components:

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

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

    const distributor& ownership() const { return get_space().ownership(); }
    const communicator& comm() const { return ownership().comm(); }
    size_type     ndof() const { return ownership().size(); }
    size_type dis_ndof() const { return ownership().dis_size(); }
          T& dof (size_type idof);
    const T& dof (size_type idof) const;
    const T& dis_dof (size_type dis_idof) const;
    iterator begin_dof();
    iterator end_dof();
    const_iterator begin_dof() const;
    const_iterator end_dof() const;
```

```
// input/output:

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

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

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

// internals:
public:

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

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

protected:
    void dis_dof_update_internal() const;
    void dis_dof_update_needed() const;

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

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

template <class T, class M>
field_basic<T,M> norm (const field_basic<T,M>& uh);

template <class T, class M>
field_basic<T,M> norm2 (const field_basic<T,M>& uh);

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

5.5 `form` - representation of a finite element bilinear form

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

Description

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

```
a: U*V   ----> IR
  (u,v)  |---> a(u,v)
```

The operator `A` associated to the bilinear form is defined by:

```
A: U  ----> V'
   u  |---> A(u)
```

where `u` and `v` are fields (see Section 5.4 [field class], page 37), and `A(u)` is such that `a(u,v)=<A(u),v>` for all u in U and v in V and where `<.,.>` denotes the duality product between V and V'. Since V is a finite dimensional spaces, the duality product is the euclidian product in IR^dim(V).

Since both U and V are finite dimensional spaces, the linear operator can be represented by a matrix. The `form` class is represented by four sparse matrix in `csr` format (see Section 5.13 [csr class], page 73), associated to unknown and blocked degrees of freedom of origin and destination spaces (see Section 5.7 [space class], page 53).

Example

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

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

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

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

Algabra

Forms, as matrices (see Section 5.13 [csr class], page 73), support linear algebra: Adding or substracting two forms writes `a+b` and `a-b`, respectively, and multiplying a form by a field `uh` writes `a*uh`. Thus, any linear combination of forms is available.

Implementation

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

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

// allocator/deallocator:

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

    form_basic (const space_type& X, const space_type& Y,
        const std::string& name = "",
        quadrature_option_type  qopt = quadrature_option_type(quadrature_option_typ

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

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

    form_basic (const space_type& X, const space_type& Y,
        const std::string& name,
        const band_basic<T,M>& bh,
        quadrature_option_type  qopt = quadrature_option_type(quadrature_option_typ

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

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

// accessors:

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

```
    const communicator& comm() const;

// linear algebra:

    form_basic<T,M>  operator+  (const form_basic<T,M>& b) const;
    form_basic<T,M>  operator-  (const form_basic<T,M>& b) const;
    form_basic<T,M>& operator*= (const T& lambda);
    field_basic<T,M> operator*  (const field_basic<T,M>& xh) const;
#ifdef TO_CLEAN
    template <class Expr>
    field_basic<T,M> operator*  (const field_expr<Expr>& xh) const;
#endif // TO_CLEAN
    field_basic<T,M> trans_mult (const field_basic<T,M>& yh) const;

    float_type operator () (const field_basic<T,M>& uh, const field_basic<T,M>& vh)

// io:

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

// accessors & modifiers to unknown & blocked parts:

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

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

// internals:
    void assembly (const form_element<T,M>&   form_e,
                   const geo_basic<T,M>&      X_geo,
                   const geo_basic<T,M>&      Y_geo,
                   bool  X_geo_is_background = true);
    void form_init (
                   const std::string&      name,
```

```
                    bool                    has_weight,
                    const field_basic<T,M>& weight,
                    quadrature_option_type  qopt);

};
template<class T, class M> form_basic<T,M> trans (const form_basic<T,M>& a);
typedef form_basic<Float,rheo_default_memory_model> form;
```

5.6 geo - finite element mesh

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

Synopsys

Distributed finite element mesh.

Implementation

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

// typedefs:

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

// allocators:

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

    // build from_list (for level set)
    geo_basic (
      const geo_basic<T,sequential>&                    lambda,
```

```
    const array<point_basic<T>,sequential>&             node_list,
    const boost::array<array<geo_element_auto<heap_allocator<size_type> >,sequent
                       reference_element::max_variant>& elt_list)
  : base (new_macro(rep_geo_rep(lambda,node_list,elt_list))) {}

// accessors:

  std::string                    name() const { return base::data().name(); }
  std::string              familyname() const { return base::data().familyname();
  size_type                 dimension() const { return base::data().dimension(); }
  size_type             map_dimension() const { return base::data().map_dimension
  size_type             serial_number() const { return base::data().serial_number
  coordinate_type   coordinate_system() const { return base::data().coordinate_sy
  std::string  coordinate_system_name() const { return space_constant::coordinate_
  const basis_basic<T>& get_piola_basis() const { return base::data().get_piola_ba
  size_type                     order() const { return base::data().get_piola_basi
  const node_type&               xmin() const { return base::data().xmin(); }
  const node_type&               xmax() const { return base::data().xmax(); }
  const distributor& geo_element_ownership(size_type dim) const { return base::dat
  const geo_size&      sizes()             const { return base::data().sizes(); }
  const geo_size&  ios_sizes()             const { return base::data().ios_sizes()
  const_reference get_geo_element (size_type dim, size_type ige) const { return ba
        reference get_geo_element (size_type dim, size_type ige)       { return ba
  size_type        n_node()   const { return base::data().n_node(); }
  const node_type&     node(size_type     inod) const { return base::data().node(i
  const node_type& dis_node(size_type dis_inod) const { return base::data().dis_no
  void dis_inod (const geo_element& K, std::vector<size_type>& dis_inod) const {
              return base::data().dis_inod(K,dis_inod); }
  node_type piola (const geo_element& K, const node_type& hat_x) const { return ba
  const array<node_type,sequential>& get_nodes() const { return base::data().get_n
  size_type dis_inod2dis_iv (size_type dis_inod) const { return base::data().dis_i

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

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

  size_type seq_locate (
              const point_basic<T>& x,
              size_type dis_ie_guest = std::numeric_limits<size_type>::max()) cons
              { return base::data().seq_locate (x, dis_ie_guest); }
```

```
    size_type dis_locate (
                const point_basic<T>& x,
                size_type dis_ie_guest = std::numeric_limits<size_type>::max()) cons
                { return base::data().dis_locate (x, dis_ie_guest); }
    void locate (
                const array<point_basic<T>, sequential>& x,
                array<size_type, sequential>& dis_ie) const
                { return base::data().locate (x, dis_ie); }
    size_type seq_trace_move (
                const point_basic<T>&     x,
                const point_basic<T>&     v,
                      point_basic<T>&     y) const
                                    { return base::data().seq_trace_move (x,v,y
    size_type dis_trace_move (
                const point_basic<T>&     x,
                const point_basic<T>&     v,
                      point_basic<T>&     y) const
                                    { return base::data().dis_trace_move (x,v,y
    void trace_ray_boundary (
                const array<point_basic<T>,sequential>&     x,
                const array<point_basic<T>,sequential>&     v,
                      array<size_type, sequential>&         dis_ie,
                      array<point_basic<T>,sequential>&     y) const
                                    { return base::data().trace_ray_boundary (x
    void trace_move (
                const array<point_basic<T>,sequential>&     x,
                const array<point_basic<T>,sequential>&     v,
                      array<size_type, sequential>&         dis_ie,
                      array<point_basic<T>,sequential>&     y) const
                                    { return base::data().trace_move (x,v,dis_i

// modifiers:

    void set_name (std::string name);
    void set_dimension (size_type dim);
    void set_serial_number (size_type i);
    void reset_order (size_type order);
    void set_coordinate_system (coordinate_type sys_coord);
    void set_coordinate_system (std::string sys_coord_name) { set_coordinate_system
    void set_nodes (const array<node_type,sequential>& x);
    void build_from_data (
        const geo_header&                                                   hdr,
        const array<node_type, sequential>&                                 node,
              boost::array<array<geo_element_auto<heap_allocator<size_type> >,sequer
                                                                            tmp_geo_ele
        bool do_upgrade);

// extended accessors:
```

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

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

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

// for compatibility with distributed interface:

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

// comparator:

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

// i/o:

    idistream& get (idistream& ips);
    odistream& put (odistream& ops) const;
    void save (std::string filename = "") const;
    void dump (std::string name) const {        base::data().dump (name); }
    bool check (bool verbose = true) const { return base::data().check(verbose); }
};
```

Implementation

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

// typedefs:
```

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

// allocators:

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

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

// accessors:

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

```
            { return base::data().get_geo_element (dim, ige); }
const_reference dis_get_geo_element (size_type dim, size_type dis_ige) const
            { return base::data().dis_get_geo_element (dim, dis_ige); }
distributor geo_element_ios_ownership (size_type dim) const {
    return base::data().geo_element_ios_ownership (dim); }
size_type ige2ios_dis_ige (size_type dim, size_type ige) const {
    return base::data().ige2ios_dis_ige (dim,ige); }
size_type dis_ige2ios_dis_ige (size_type dim, size_type dis_ige) const {
    return base::data().dis_ige2ios_dis_ige (dim,dis_ige); }
size_type ios_ige2dis_ige (size_type dim, size_type ios_ige) const {
    return base::data().ios_ige2dis_ige (dim, ios_ige); }
size_type        n_node() const { return base::data().n_node(); }
const node_type&     node(size_type     inod) const { return base::data().node(
const node_type& dis_node(size_type dis_inod) const { return base::data().dis_n
void dis_inod (const geo_element& K, std::vector<size_type>& dis_inod) const {
            return base::data().dis_inod(K,dis_inod); }
node_type piola (const geo_element& K, const node_type& hat_x) const { return ba
const array<node_type,distributed>& get_nodes() const { return base::data().get_
void set_nodes (const array<node_type,distributed>& x);
void reset_order (size_type order);
size_type dis_inod2dis_iv (size_type dis_inod) const { return base::data().dis_i
void set_coordinate_system (coordinate_type sys_coord);
void set_coordinate_system (std::string sys_coord_name) { set_coordinate_system
void set_dimension (size_type dim);
void set_serial_number (size_type i);
void set_name (std::string name);

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

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

size_type seq_locate (
            const point_basic<T>& x,
            size_type dis_ie_guest = std::numeric_limits<size_type>::max()) cons
            { return base::data().seq_locate (x, dis_ie_guest); }
size_type dis_locate (
            const point_basic<T>& x,
            size_type dis_ie_guest = std::numeric_limits<size_type>::max()) cons
            { return base::data().dis_locate (x, dis_ie_guest); }
void locate (const array<point_basic<T>, distributed>& x, array<size_type, dist
```

```
            { return base::data().locate (x, dis_ie); }
    size_type seq_trace_move (
            const point_basic<T>&     x,
            const point_basic<T>&     v,
                  point_basic<T>&     y) const
                                    { return base::data().seq_trace_move (x,v,y
    size_type dis_trace_move (
            const point_basic<T>&     x,
            const point_basic<T>&     v,
                  point_basic<T>&     y) const
                                    { return base::data().dis_trace_move (x,v,y
    void trace_ray_boundary (
            const array<point_basic<T>,distributed>&     x,
            const array<point_basic<T>,distributed>&     v,
                  array<size_type, distributed>&         dis_ie,
                  array<point_basic<T>,distributed>&     y) const
                                    { return base::data().trace_ray_boundary (x
    void trace_move (
            const array<point_basic<T>,distributed>&     x,
            const array<point_basic<T>,distributed>&     v,
                  array<size_type, distributed>&         dis_ie,
                  array<point_basic<T>,distributed>&     y) const
                                    { return base::data().trace_move (x,v,dis_i

// extended accessors:

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

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

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

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

// comparator:
```

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

// i/o:

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

// utilities:

    void set_ios_permutation (
        boost::array<size_type,reference_element::max_variant>& loc_ndof_by_variant
        array<size_type,distributed>&                           idof2ios_dis_idof) c
     { base::data().set_ios_permutation (loc_ndof_by_variant, idof2ios_dis_idof); }
};
```

5.7 space – piecewise polynomial finite element space

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

Description

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

Synopsis

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

Product

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

Implementation

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

// typedefs:

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

```
    typedef typename rep::valued_type valued_type;

// allocators:

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

// accessors:

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

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

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

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

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

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

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

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

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

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

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

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

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

// comparator:

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

Implementation

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

// typedefs:

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

// allocators:

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

// accessors:

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

```
void unblock(const domain_indirect_basic<distributed>& dom);

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

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

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

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

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

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

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

const point_basic<T>& xdof (size_type idof) const;
const array<point_basic<T>,distributed>& get_xdofs() const;

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

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

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

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

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

// comparator:

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

5.8 point - vertex of a mesh

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

Description

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

Implementation

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

// typedefs:

        typedef size_t size_type;
        typedef T      float_type;

// allocators:

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

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

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

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

```
// accessors:

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

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

// inputs/outputs:

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

// algebra:

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

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

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

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

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

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

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

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

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

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

        point_basic<T> operator/ (const T& a) const
                { return operator* (T(1)/T(a)); }

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

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

// algebra:
template<class T>
inline
point_basic<T>
operator* (int a, const point_basic<T>& u)
{
  return u.operator* (a);
}
template<class T>
inline
point_basic<T>
operator* (const T& a, const point_basic<T>& u)
{
  return u.operator* (a);
}
template<class T>
inline
point_basic<T>
vect (const point_basic<T>& v, const point_basic<T>& w)
{
  return point_basic<T> (
```

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

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

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

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

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

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

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

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

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

(Source file: 'nfem/geo_element/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

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

        typedef size_t size_type;
        typedef T  element_type;

// allocators:

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

// affectation:

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

// modifiers:

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

// accessors:

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

// inputs/outputs:

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

```
// algebra:

        bool operator== (const tensor_basic<T>&) const;
        bool operator!= (const tensor_basic<T>& b) const { return ! operator== (b);
        template <class U>
        friend tensor_basic<U> operator- (const tensor_basic<U>&);
        template <class U>
        friend tensor_basic<U> operator+ (const tensor_basic<U>&, const tensor_basi
        template <class U>
        friend tensor_basic<U> operator- (const tensor_basic<U>&, const tensor_basi
        template <class U>
        friend tensor_basic<U> operator* (int k, const tensor_basic<U>& a);
        template <class U>
        friend tensor_basic<U> operator* (const U& k, const tensor_basic<U>& a);
        template <class U>
        friend tensor_basic<U> operator* (const tensor_basic<U>& a, int k);
        template <class U>
        friend tensor_basic<U> operator* (const tensor_basic<U>& a, const U& k);
        template <class U>
        friend tensor_basic<U> operator/ (const tensor_basic<U>& a, int k);
        template <class U>
        friend tensor_basic<U> operator/ (const tensor_basic<U>& a, const U& k);
        template <class U>
        friend point_basic<U>  operator* (const tensor_basic<U>&, const point_basic
        template <class U>
        friend point_basic<U>  operator* (const point_basic<U>& yt, const tensor_ba
               point_basic<T>  trans_mult (const point_basic<T>& x) const;
        template <class U>
        friend tensor_basic<U> trans      (const tensor_basic<U>& a, size_t d = 3);
        template <class U>
        friend tensor_basic<U> operator* (const tensor_basic<U>& a, const tensor_ba
        template <class U>
        friend void prod (const tensor_basic<U>& a, const tensor_basic<U>& b, tenso
                size_t di=3, size_t dj=3, size_t dk=3);
        template <class U>
        friend tensor_basic<U> inv        (const tensor_basic<U>& a, size_t d = 3);
        template <class U>
        friend tensor_basic<U> diag (const point_basic<U>& d);
        template <class U>
        friend tensor_basic<U> identity (size_t d=3);
        template <class U>
        friend tensor_basic<U> dyadic (const point_basic<U>& u, const point_basic<U

// metric and geometric transformations:

        template <class U>
        friend U dotdot (const tensor_basic<U>&, const tensor_basic<U>&);
        template <class U>
        friend U norm2 (const tensor_basic<U>& a) { return dotdot(a,a); }
```

```
        template <class U>
        friend U dist2 (const tensor_basic<U>& a, const tensor_basic<U>& b) { retur
        template <class U>
        friend U norm  (const tensor_basic<U>& a) { return ::sqrt(norm2(a)); }
        template <class U>
        friend U dist  (const tensor_basic<U>& a, const tensor_basic<U>& b) { retur
        T determinant (size_type d = 3) const;
        template <class U>
        friend U determinant (const tensor_basic<U>& A, size_t d = 3);
        template <class U>
        friend bool invert_3x3 (const tensor_basic<U>& A, tensor_basic<U>& 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_basic<T> eig (tensor_basic<T>& q, size_t dim = 3) const;
        point_basic<T> eig (size_t dim = 3) const;

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

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

// inputs/outputs:
template<class T>
inline
std::istream& operator>> (std::istream& in, tensor_basic<T>& a)
{
    return a.get (in);
}
template<class T>
inline
std::ostream& operator<< (std::ostream& out, const tensor_basic<T>& a)
{
    return a.put (out);
}
```

```
// t = a otimes b
template<class T>
tensor_basic<T> otimes (const point_basic<T>& a, const point_basic<T>& b, size_t na
// t += a otimes b
template<class T>
void cumul_otimes (tensor_basic<T>& t, const point_basic<T>& a, const point_basic<T
template<class T>
void cumul_otimes (tensor_basic<T>& t, const point_basic<T>& a, const point_basic<T
```

5.10 `tensor4` - a fouth order tensor

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

Synopsys

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

Implementation

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

  typedef size_t size_type;
  typedef T      element_type;

// allocators:

  tensor4_basic (const T& init_val = 0);
  tensor4_basic (const tensor4_basic<T>& a);

// affectation:

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

// accessors:

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

// data:
protected:
  T _x [3][3][3][3];
};
typedef tensor4_basic<Float> tensor4;
```

5.11 `array` - container in distributed environment

(Source file: '`skit/plib2/array.h`')

Synopsys

STL-like vector container for a distributed memory machine model.

Example

A sample usage of the class is:

```
int main(int argc, char**argv) {
   environment distributed(argc, argv);
   array<double> x(distributor(100), 3.14);
   dout << x << endl;
}
```

The array<T> interface is similar to those of the std::vector<T> with the addition of some communication features in the distributed case: write accesses with entry/assembly and read access with dis_at.

Distributed write access

Loop on any `dis_i` that is not managed by the current processor:

```
x.dis_entry (dis_i) = value;
```

and then, after loop, perform all communication:

```
x.dis_entry_assembly();
```

After this command, each value is stored in the array, available the processor associated to `dis_i`.

Distributed read access

First, define the set of indexes:

```
std::set<size_t> ext_idx_set;
```

Then, loop on `dis_i` indexes that are not managed by the current processor:

```
ext_idx_set.insert (dis_i);
```

After the loop, performs the communications:

```
x.set_dis_indexes (ext_idx_set);
```

After this command, each values associated to the `dis_i` index, and that belongs to the index set, is now available also on the current processor as:

```
value = x.dis_at (dis_i);
```

For convenience, if `dis_i` is managed by the current processor, this function returns also the value.

Note

The class takes two template parameters: one for the type T and the second for the memory model M, that could be either M=distributed or M=sequential. The two cases are associated to two diferent implementations, but proposes exactly the same interface. The sequential interface propose also a supplementary constructor:

```
array<double,sequential> x(local_size, init_val);
```

This constructor is a STL-like one but could be consufused in the distributed case, since there are two sizes: a local one and a global one. In that case, the use of the distributor, as a generalization of the size concept, clarify the situation (see Section 5.15 [distributor class], page 81).

Implementation note

"scatter" via "get_dis_entry".

"gather" via "dis_entry(dis_i) = value" or "dis_entry(dis_i) += value". Note that += applies when T=idx_set where idx_set is a wrapper class of std::set<size_t> ; the += operator represents the union of a set. The operator= is used when T=double or others simple T types without algebra. If there is a conflict, i.e. several processes set the dis_i index, then the result of operator+= depends upon the order of the process at each run and is not deterministic. Such ambiguous behavior is not detected yet at run time.

Implementation

```
template <class T, class A>
class array<T,sequential,A> : public smart_pointer<array_seq_rep<T,A> > {
public:

// typedefs:

    typedef array_seq_rep<T,A>            rep;
    typedef smart_pointer<rep>            base;

    typedef sequential                    memory_type;
    typedef typename rep::size_type       size_type;
    typedef typename rep::difference_type difference_type;
    typedef typename rep::value_type      value_type;
    typedef typename rep::reference       reference;
    typedef typename rep::dis_reference   dis_reference;
    typedef typename rep::iterator        iterator;
    typedef typename rep::const_reference const_reference;
    typedef typename rep::const_iterator  const_iterator;
```

```
// allocators:

    array       (size_type loc_size = 0,       const T& init_val = T(), const A& al
    void resize (size_type loc_size = 0,       const T& init_val = T());
    array       (const distributor& ownership, const T& init_val = T(), const A& al
    void resize (const distributor& ownership, const T& init_val = T());

// local accessors & modifiers:

    A get_allocator() const              { return base::data().get_allocator(); }
    size_type     size () const          { return base::data().size(); }
    size_type dis_size () const          { return base::data().dis_size(); }
    const distributor& ownership() const { return base::data().ownership(); }
    const communicator& comm() const     { return ownership().comm(); }

    reference       operator[] (size_type i)       { return base::data().operator[]
    const_reference operator[] (size_type i) const { return base::data().operator[]
    reference       operator() (size_type i)       { return base::data().operator[]
    const_reference operator() (size_type i) const { return base::data().operator[]
    const_reference dis_at (size_type dis_i) const { return operator[] (dis_i); }

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

// global modifiers (for compatibility with distributed interface):

    dis_reference dis_entry (size_type dis_i) { return base::data().dis_entry(dis_i
    void dis_entry_assembly()                 {}
    template<class SetOp> void dis_entry_assembly(SetOp my_set_op)       {}
    template<class SetOp> void dis_entry_assembly_begin (SetOp my_set_op) {}
    template<class SetOp> void dis_entry_assembly_end (SetOp my_set_op)   {}

    void reset_dis_indexes() const {}
    template<class Set> void set_dis_indexes    (const Set& ext_idx_set) const {}
    template<class Set> void append_dis_indexes (const Set& ext_idx_set) const {}
    template<class Set, class Map> void append_dis_entry (const Set& ext_idx_set, Ma
    template<class Set, class Map> void get_dis_entry    (const Set& ext_idx_set, Ma

// apply a partition:

    template<class RepSize>
    void repartition (                              // old_numbering for *this
        const RepSize&         partition,           // old_ownership
        array<T,sequential,A>& new_array,           // new_ownership (created)
        RepSize&               old_numbering,       // new_ownership
```

```
        RepSize&               new_numbering) const  // old_ownership
        { return base::data().repartition (partition, new_array, old_numbering, new

    template<class RepSize>
    void permutation_apply (                      // old_numbering for *this
        const RepSize&          new_numbering,    // old_ownership
        array<T,sequential,A>&  new_array) const  // new_ownership (already alloca
        { return base::data().permutation_apply (new_numbering, new_array); }

    void reverse_permutation (                                  // old_ownership for
        array<size_type,sequential,A>& inew2dis_iold) const   // new_ownership
        { base::data().reverse_permutation (inew2dis_iold.data()); }

// i/o:

    odiststream& put_values (odiststream& ops) const { return base::data().put_valu
    idiststream& get_values (idiststream& ips)       { return base::data().get_valu
    template <class GetFunction>
    idiststream& get_values (idiststream& ips, GetFunction get_element)       { ret
    template <class PutFunction>
    odiststream& put_values (odiststream& ops, PutFunction put_element) const { ret
    void dump (std::string name) const { return base::data().dump(name); }
};
```

Implementation

```
template <class T, class A>
class array<T,distributed,A> : public smart_pointer<array_mpi_rep<T,A> > {
public:

// typedefs:

    typedef array_mpi_rep<T,A>            rep;
    typedef smart_pointer<rep>            base;

    typedef distributed                   memory_type;
    typedef typename rep::size_type       size_type;
    typedef typename rep::difference_type difference_type;
    typedef typename rep::value_type      value_type;
    typedef typename rep::reference       reference;
    typedef typename rep::dis_reference   dis_reference;
    typedef typename rep::iterator        iterator;
    typedef typename rep::const_reference const_reference;
    typedef typename rep::const_iterator  const_iterator;
    typedef typename rep::scatter_map_type scatter_map_type;

// allocators:

    array       (const distributor& ownership = distributor(), const T& init_val =
```

```
    void resize (const distributor& ownership = distributor(), const T& init_val = T

// local accessors & modifiers:

    A get_allocator() const             { return base::data().get_allocator(); }
    size_type     size () const         { return base::data().size(); }
    size_type dis_size () const         { return base::data().dis_size(); }
    const distributor& ownership() const { return base::data().ownership(); }
    const communicator& comm() const    { return base::data().comm(); }

    reference       operator[] (size_type i)       { return base::data().operator[]
    const_reference operator[] (size_type i) const { return base::data().operator[]
    reference       operator() (size_type i)       { return base::data().operator[]
    const_reference operator() (size_type i) const { return base::data().operator[]

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

// global accessor:

    template<class Set, class Map>
    void append_dis_entry (const Set& ext_idx_set, Map& ext_idx_map) const { base::

    template<class Set, class Map>
    void get_dis_entry    (const Set& ext_idx_set, Map& ext_idx_map) const { base::

    template<class Set>
    void append_dis_indexes (const Set& ext_idx_set) const { base::data().append_dis
    void reset_dis_indexes() const { base::data().reset_dis_indexes(); }

    template<class Set>
    void set_dis_indexes    (const Set& ext_idx_set) const { base::data().set_dis_i

    const T& dis_at (size_type dis_i) const { return base::data().dis_at (dis_i); }

    // get all external pairs (dis_i, values):
    const scatter_map_type& get_dis_map_entries() const { return base::data().get_di

// global modifiers (for compatibility with distributed interface):

    dis_reference dis_entry (size_type dis_i)       { return base::data().dis_entry

    void dis_entry_assembly()                       { return base::data().dis_entry_

    template<class SetOp>
    void dis_entry_assembly       (SetOp my_set_op) { return base::data().dis_entry_
    template<class SetOp>
```

```
    void dis_entry_assembly_begin (SetOp my_set_op) { return base::data().dis_entry
    template<class SetOp>
    void dis_entry_assembly_end   (SetOp my_set_op) { return base::data().dis_entry

// apply a partition:

    template<class RepSize>
    void repartition (                              // old_numbering for *this
        const RepSize&        partition,            // old_ownership
        array<T,distributed>& new_array,            // new_ownership (created)
        RepSize&              old_numbering,        // new_ownership
        RepSize&              new_numbering) const  // old_ownership
        { return base::data().repartition (partition.data(), new_array.data(), old_

    template<class RepSize>
    void permutation_apply (                       // old_numbering for *this
        const RepSize&          new_numbering,     // old_ownership
        array<T,distributed,A>& new_array) const   // new_ownership (already alloca
        { base::data().permutation_apply (new_numbering.data(), new_array.data()); }

    void reverse_permutation (                                  // old_ownership for
        array<size_type,distributed,A>& inew2dis_iold) const   // new_ownership
        { base::data().reverse_permutation (inew2dis_iold.data()); }

// i/o:

    odiststream& put_values (odiststream& ops) const { return base::data().put_valu
    idiststream& get_values (idiststream& ips)       { return base::data().get_valu
    void dump (std::string name) const      { return base::data().dump(name); }

    template <class GetFunction>
    idiststream& get_values (idiststream& ips, GetFunction get_element)       { ret
    template <class PutFunction>
    odiststream& put_values (odiststream& ops, PutFunction put_element) const { ret
    template <class PutFunction, class A2> odiststream& permuted_put_values (
                odiststream& ops, const array<size_type,distributed,A2>& perm, PutF
                                                                        { return base:
};
```

5.12 asr - associative sparse matrix

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

Synopsys

Associative sparse matrix container stored row by row using the STL map class.

Implementation note

Implementation use MPI-1.1 and is inspired from Mat_MPI in PETSc-2.0.22.

To do

For efficiency purpose, the assembly phase may access directly to the asr representation, without crossing the reference counting and pointer handler. Something like the iterator for dense vectors.

Implementation

```
template<class R>
class basic_asr : public smart_pointer<R> {
public:

// typedefs:

    typedef typename R::size_type          size_type;
    typedef typename R::element_type       element_type;
    typedef typename R::memory_type        memory_type;
    typedef distributor::communicator_type communicator_type;

// allocators/deallocators:

    basic_asr (size_type dis_nrow = 0, size_type dis_ncol = 0);
    basic_asr (const distributor& row_ownership, const distributor& col_ownership);
    explicit basic_asr (const csr<element_type,memory_type>&);

// accessors:

    const communicator_type& comm() const;

    // local sizes
    size_type nrow () const;
    size_type ncol () const;
    size_type nnz () const;

    // global sizes
    size_type dis_nrow () const;
    size_type dis_ncol () const;
    size_type dis_nnz () const;
    const distributor& row_ownership() const;
    const distributor& col_ownership() const;

    // range on local memory
    size_type row_first_index () const;
    size_type row_last_index () const;
    size_type col_first_index () const;
    size_type col_last_index () const;

// global modifiers:

    element_type& dis_entry (size_type dis_i, size_type dis_j);
```

```
    void dis_entry_assembly();
    void dis_entry_assembly_begin ();
    void dis_entry_assembly_end ();
    void resize (size_type dis_nrow = 0, size_type dis_ncol = 0);

// output:

    void dump (const std::string& name) const;
};
template <class T, class M = rheo_default_memory_model>
class asr
{
    typedef M memory_type;
};
template <class T>
class asr<T,sequential> : public basic_asr<asr_seq_rep<T> > {
public:
    typedef typename basic_asr<asr_seq_rep<T> >::size_type size_type;
    typedef sequential memory_type;
    asr (size_type dis_nrow = 0, size_type dis_ncol = 0);
    asr (const distributor& row_ownership, const distributor& col_ownertship);
    explicit asr(const csr<T,memory_type>&);
};
#ifdef _RHEOLEF_HAVE_MPI
template <class T>
class asr<T,distributed> : public basic_asr<asr_mpi_rep<T> > {
public:
    typedef distributed memory_type;
    typedef typename basic_asr<asr_mpi_rep<T> >::size_type size_type;
    asr (size_type dis_nrow = 0, size_type dis_ncol = 0);
    asr (const distributor& row_ownership, const distributor& col_ownertship);
    explicit asr(const csr<T,memory_type>&);
};
#endif // _RHEOLEF_HAVE_MPI

// inputs/outputs:
template <class T, class M>
idiststream& operator >> (idiststream& s, asr<T,M>& x);

template <class T, class M>
odiststream& operator << (odiststream& s, const asr<T,M>& x);
```

5.13 csr - compressed sparse row matrix

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

Synopsys

Distributed compressed sparse matrix container stored row by row.

Description

Sparse matrix are compressed by rows. In distributed environment, the distribution follows the row distributor (see Section 5.15 [distributor class], page 81).

Algebra

Adding or substracting two matrices writes `a+b` and `a-b`, respectively, and multiplying a matrix by a scalar writes `lambda*x`. Thus, any linear combination of sparse matrices is available.

Matrix-vector product writes `a*x` where `x` is a vector (see Section 5.19 [vec class], page 86).

Limitations

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

Implementation

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

// typedefs:

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

// allocators/deallocators:

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

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

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

```
#endif // _RHEOLEF_HAVE_STD_INITIALIZER_LIST

// accessors:

    // global sizes
    const distributor& row_ownership() const { return base::data().row_ownership();
    const distributor& col_ownership() const { return base::data().col_ownership();
    size_type dis_nrow () const              { return row_ownership().dis_size(); }
    size_type dis_ncol () const              { return col_ownership().dis_size(); }
    size_type dis_nnz () const               { return base::data().nnz(); }
    bool is_symmetric() const                { return base::data().is_symmetric(); }
    void set_symmetry (bool is_symm)         { base::data().set_symmetry(is_symm); }
    size_type pattern_dimension() const      { return base::data().pattern_dimensio
    void set_pattern_dimension(size_type dim){ base::data().set_pattern_dimension(di

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

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

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

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

// algebra:

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

```
      base::data().trans_mult (x,y);
    }
    vec<element_type,sequential> trans_mult (const vec<element_type,sequential>& x)
      vec<element_type,sequential> y (col_ownership(), element_type());
      trans_mult (x, y);
      return y;
    }
    // a+b, a-b
    csr<T,sequential> operator+ (const csr<T,sequential>& b) const;
    csr<T,sequential> operator- (const csr<T,sequential>& b) const;

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

    void dump (const std::string& name) const { base::data().dump(name); }
};
// lambda*a
template<class T>
inline
csr<T,sequential>
operator* (const T& lambda, const csr<T,sequential>& a)
{
  csr<T,sequential> b = a;
  b.operator*= (lambda);
  return b;
}
// -a
template<class T>
inline
csr<T,sequential>
operator- (const csr<T,sequential>& a)
{
  return T(-1)*a;
}
// trans(a)
template<class T>
inline
csr<T,sequential>
trans (const csr<T,sequential>& a)
{
  csr<T,sequential> b;
  a.data().build_transpose (b.data());
  return b;
}
```

Implementation

```
template<class T>
class csr<T,distributed> : public smart_pointer<csr_mpi_rep<T> > {
public:

// typedefs:

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

// allocators/deallocators:

    csr() : base(new_macro(rep())) {}
    explicit csr(const asr<T,memory_type>& a) : base(new_macro(rep(a.data()))) {}
    void resize (const distributor& row_ownership, const distributor& col_ownership
        { base::data().resize(row_ownership, col_ownership, nnz1); }

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

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

// accessors:

    // global sizes
    const distributor& row_ownership() const { return base::data().row_ownership();
    const distributor& col_ownership() const { return base::data().col_ownership();
    size_type dis_nrow () const              { return row_ownership().dis_size(); }
    size_type dis_ncol () const              { return col_ownership().dis_size(); }
    size_type dis_nnz () const               { return base::data().dis_nnz(); }
    bool is_symmetric() const                { return base::data().is_symmetric(); }
    void set_symmetry (bool is_symm)         { base::data().set_symmetry(is_symm); }
    size_type pattern_dimension() const      { return base::data().pattern_dimensio
    void set_pattern_dimension(size_type dim){ base::data().set_pattern_dimension(di

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

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

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

    // accessors, only for distributed
    size_type ext_nnz() const                { return base::data().ext_nnz(); }
    const_iterator ext_begin() const         { return base::data().ext_begin(); }
    const_iterator ext_end()   const         { return base::data().ext_end(); }
    size_type jext2dis_j (size_type jext) const { return base::data().jext2dis_j(je

// algebra:


    // y := a*x
    void mult (const vec<element_type,distributed>& x, vec<element_type,distributed
      base::data().mult (x,y);
    }
    vec<element_type,distributed> operator* (const vec<element_type,distributed>& x
      vec<element_type,distributed> y (row_ownership(), element_type());
      mult (x, y);
      return y;
    }
    void trans_mult (const vec<element_type,distributed>& x, vec<element_type,distri
      base::data().trans_mult (x,y);
    }
    vec<element_type,distributed> trans_mult (const vec<element_type,distributed>& x
      vec<element_type,distributed> y (col_ownership(), element_type());
      trans_mult (x, y);
      return y;
    }
    // a+b, a-b
    csr<T,distributed> operator+ (const csr<T,distributed>& b) const;
    csr<T,distributed> operator- (const csr<T,distributed>& b) const;

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

```
    void dump (const std::string& name) const { base::data().dump(name); }
};
// lambda*a
template<class T>
inline
csr<T,distributed>
operator* (const T& lambda, const csr<T,distributed>& a)
{
  csr<T,distributed> b = a;
  b.operator*= (lambda);
  return b;
}
// -a
template<class T>
inline
csr<T,distributed>
operator- (const csr<T,distributed>& a)
{
  return T(-1)*a;
}
// trans(a)
template<class T>
inline
csr<T,distributed>
trans (const csr<T,distributed>& a)
{
  csr<T,distributed> b;
  a.data().build_transpose (b.data());
  return b;
}
```

5.14 `dia` - diagonal matrix

(Source file: '`skit/plib2/dia.h`')

Description

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

```
dia<Float> d;
```

The constructor can be invocated whith a `ownership` parameter (see Section 5.15 [distributor class], page 81):

```
dia<Float> d(ownership);
```

or an initialiser, either a vector (see Section 5.19 [vec class], page 86):

```
dia<Float> d(v);
```

or a csr matrix (see Section 5.13 [csr class], page 73):

```
dia<Float> d(a);
```

The conversion from `dia` to `vec` or `csr` is explicit.

When a diagonal matrix is constructed from a `csr` matrix, the definition of the diagonal of matrix is *always* a vector of size *row_ownership* which contains the elements in rows 1 to *nrow* of the matrix that are contained in the diagonal. If the diagonal element falls outside the matrix, i.e. *ncol* < *nrow* then it is defined as a zero entry.

Preconditioner interface

The class presents a preconditioner interface, as the Section 5.17 [solver class], page 83, so that it can be used as preconditioner to the iterative solvers suite (see Section 7.10 [pcg algorithm], page 112).

Implementation

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

// typedefs:

    typedef typename vec<T,M>::size_type       size_type;
    typedef typename vec<T,M>::iterator          iterator;
    typedef typename vec<T,M>::const_iterator    const_iterator;

// allocators/deallocators:

    explicit dia (const distributor& ownership = distributor(),
                  const T&  init_val = std::numeric_limits<T>::max());

    explicit dia (const vec<T,M>& u);
    explicit dia (const csr<T,M>& a);
    dia<T,M>& operator= (const T& lambda);

// preconditionner interface: solves d*x=b

    vec<T,M> solve (const vec<T,M>& b) const;
    vec<T,M> trans_solve (const vec<T,M>& b) const;
};
template <class T, class M>
dia<T,M> operator/ (const T& lambda, const dia<T,M>& d);

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

5.15 `distributor` - data distribution table

(Source file: '`skit/plib2/distributor.h`')

Synopsys

Used by "array"(1), "asr"(1) and "csr"(1). and such classes that distribute data as chunk.

Implementation

```
class distributor : public Vector<std::allocator<int>::size_type> {
public:

// typedefs:

        typedef std::allocator<int>::size_type size_type;
        typedef Vector<size_type>              _base;
        typedef _base::iterator                iterator;
        typedef _base::const_iterator          const_iterator;
        typedef int                            tag_type;
        typedef communicator                   communicator_type;

// constants:

        static const size_type decide = size_type(-1);

// allocators/deallocators:

        distributor(
                size_type dis_size = 0,
                const communicator_type& c = communicator_type(),
                size_type loc_size = decide);

        distributor(const distributor&);
        ~distributor();

        void resize(
                size_type dis_size = 0,
                const communicator_type& c = communicator_type(),
                size_type loc_size = decide);

// accessors:

        const communicator_type& comm() const;

        /// global and local sizes
        size_type dis_size () const;

        /// current process id
        size_type process () const;
```

```
        /// number of processes
        size_type n_process () const;

        /// find iproc associated to a global index dis_i: CPU=log(nproc)
        size_type find_owner (size_type dis_i) const;

        /// global index range and local size owned by ip-th process
        size_type first_index (size_type ip) const;
        size_type last_index (size_type ip) const;
        size_type size (size_type ip) const;

        /// global index range and local size owned by current process
        size_type first_index () const;
        size_type last_index () const;
        size_type size () const;

        /// true when dis_i in [first_index(ip):last_index(ip)[
        bool is_owned (size_type dis_i, size_type ip) const;

        // the same with ip=current process
        bool is_owned (size_type dis_i) const;


        /// returns a new tag
        static tag_type get_new_tag();

// comparators:

        bool operator== (const distributor&) const;
        bool operator!= (const distributor&) const;
// data:
protected:
        communicator_type _comm;
};
```

5.16 eye - the identity matrix

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

Description

Following matlab, the name `eye()` is used in place of I to denote identity matrices because I is often used as a subscript or as sqrt(-1). The dimensions of `eye()` are determined by context. The preconditioner interface is usefull when calling algorithms without any preconditioners, e.g.

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

Implementation

```
class eye {
public:
  eye() {}
  template<class T> const vec<T>& operator* (const vec<T>& x) const { return x; }
  template<class T> const vec<T>& solve (const vec<T>& x) const { return x; }
  template<class T> const vec<T>& trans_solve (const vec<T>& x) const { x; }
};
```

5.17 `solver` - direct or interative solver interface

(Source file: '`skit/plib2/solver.h`')

Description

The class implements a matrix factorization: LU factorization for an unsymmetric matrix and Choleski fatorisation for a symmetric one.

Let *a* be a square invertible matrix in `csr` format (see Section 5.13 [csr class], page 73).

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

We get the factorization by:

```
solver<Float> sa (a);
```

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

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

When the matrix is modified in a computation loop but conserves its sparsity pattern, an efficient re-factorization writes:

```
sa.update_values (new_a);
x = sa.solve(b);
```

This approach skip the long step of the symbolic factization step.

Iterative solver

The factorization can also be incomplete, i.e. a pseudo-inverse, suitable for preconditionning iterative methods. In that case, the sa.solve(b) call runs a conjugate gradient when the matrix is symmetric, or a generalized minimum residual algorithm when the matrix is unsymmetric.

Automatic choice and customization

The symmetry of the matrix is tested via the a.is_symmetric() property (see Section 5.13 [csr class], page 73) while the choice between direct or iterative solver is switched from the a.pattern_dimension() value. When the pattern is 3D, an iterative method is faster and less memory consuming. Otherwhise, for 1D or 2D problems, the direct method is prefered.

These default choices can be supersetted by using explicit options:

```
solver_option_type opt;
opt.iterative = true;
solver<Float> sa (a, opt);
```

See the solver.h header for the complete list of available options.

Implementation note

The implementation bases on the pastix library.

Implementation

```
template <class T, class M = rheo_default_memory_model>
class solver_basic : public smart_pointer<solver_rep<T,M> > {
public:
// typedefs:

  typedef solver_rep<T,M>    rep;
  typedef smart_pointer<rep> base;

// allocator:

  solver_basic ();
  explicit solver_basic (const csr<T,M>& a, const solver_option_type& opt = solver_
  void update_values (const csr<T,M>& a);

// accessors:

  vec<T,M> trans_solve (const vec<T,M>& b) const;
  vec<T,M> solve       (const vec<T,M>& b) const;
};
// factorizations:
template <class T, class M>
solver_basic<T,M> ldlt(const csr<T,M>& a, const solver_option_type& opt = solver_op
template <class T, class M>
solver_basic<T,M> lu  (const csr<T,M>& a, const solver_option_type& opt = solver_op
template <class T, class M>
solver_basic<T,M> ic0 (const csr<T,M>& a, const solver_option_type& opt = solver_op
template <class T, class M>
solver_basic<T,M> ilu0(const csr<T,M>& a, const solver_option_type& opt = solver_op

typedef solver_basic<Float> solver;
```

5.18 solver_abtb – direct or iterative solver iterface for mixed linear systems

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

Synopsis

```
solver_abtb stokes     (a,b,mp);
solver_abtb elasticity (a,b,c,mp);
```

Description

The `solver_abtb` class provides direct or iterative algorithms for some mixed problem:

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

where A is symmetric positive definite and C is symmetric positive. By default, iterative algorithms are considered for tridimensional problems and direct methods otherwise. Such mixed linear problems appears for instance with the discretization of Stokes problems. The C matrix can be zero and then the corresponding argument can be omitted when invoking the constructor. Non-zero C matrix appears for of Stokes problems with stabilized P1-P1 element, or for nearly incompressible elasticity problems.

Direct algorithm

When the kernel of `B^T` is not reduced to zero, then the pressure p is defined up to a constant and the system is singular. In the case of iterative methods, this is not a problem. But when using direct method, the system is then completed to impose a constraint on the pressure term and the whole matrix is factored one time for all.

Iterative algorithm

The preconditionned conjugate gradient algorithm is used, where the `mp` matrix is used as preconditionner. See see Section 7.9 [mixed_solver algorithm], page 111.

Examples

See the user's manual for practical examples for the nearly incompressible elasticity, the Stokes and the Navier-Stokes problems.

Implementation

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

// typedefs:

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

// allocators:
```

```
  solver_abtb_basic ();
  solver_abtb_basic (const csr<T,M>& a, const csr<T,M>& b, const csr<T,M>& mp,
       const solver_option_type& opt = solver_option_type());
  solver_abtb_basic (const csr<T,M>& a, const csr<T,M>& b, const csr<T,M>& c, const
       const solver_option_type& opt = solver_option_type());

// accessors:

  void solve (const vec<T,M>& f, const vec<T,M>& g, vec<T,M>& u, vec<T,M>& p) const

protected:
// internal
  void init();
// data:
  mutable solver_option_type _opt;
  csr<T,M>          _a;
  csr<T,M>          _b;
  csr<T,M>          _c;
  csr<T,M>          _mp;
  solver_basic<T,M> _sA;
  solver_basic<T,M> _sa;
  solver_basic<T,M> _smp;
  bool              _need_constraint;
};
typedef solver_abtb_basic<Float,rheo_default_memory_model> solver_abtb;
```

5.19 vec - vector in distributed environment

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

Synopsys

STL-like vector container for a sequential or distributed memory machine model. Additional operation fom classical algebra.

Example

A sample usage of the class is:

```
int main(int argc, char**argv) {
   environment distributed(argc, argv);
   vec<double> x(100, 3.14);
   dout << x << endl;
}
```

Implementation note

Implementation use array<T,M>.

Implementation

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

// typedef:

    typedef array<T, M> base;
    typedef typename base::size_type                          size_type;
    typedef std::ptrdiff_t                                    difference_type;
#ifdef TODO
    // pb compile avec boost sur foehn:
    typedef typename base::difference_type                    difference_type;
#endif // TODO
    typedef basic_range<size_type, difference_type>           range_type;
    typedef typename base::reference                          reference;
    typedef typename base::const_reference                    const_reference;
    typedef typename base::iterator                           iterator;
    typedef typename base::const_iterator                     const_iterator;

// allocator/deallocator:

    vec (const distributor& ownership,
        const T&  init_val = std::numeric_limits<T>::max());

    vec(size_type dis_size = 0,
        const T&  init_val = std::numeric_limits<T>::max());

    void resize (
        const distributor& ownership,
        const T&  init_val = std::numeric_limits<T>::max());

    void resize (
        size_type size = 0,
        const T&  init_val = std::numeric_limits<T>::max());

// accessors:

    const_reference operator[] (size_type i) const;
    reference       operator[] (size_type i);

    T max_abs () const;

// range:

    vec(const vec_range<T,M>& vr);
    vec(const vec_range_const<T,M>& vr);
    vec<T,M>& operator= (const vec_range<T,M>& vr);
```

```
    vec<T,M>& operator= (const vec_range_const<T,M>& vr);

    vec_range_const<T,M> operator[] (const range_type& r) const;
    vec_range<T,M>       operator[] (const range_type& r);

// assignment to a constant:

    vec<T,M>& operator= (const int& expr);
    vec<T,M>& operator= (const T& expr);

// expression template:

    template<typename Expr>
    vec (const Expr& expr);

    template<typename Expr>
    vec<T,M>& operator= (const vec_expr<Expr>& expr);

    template<typename Expr>
    vec<T,M>& operator+= (const Expr& expr);

    template<typename Expr>
    vec<T,M>& operator-= (const Expr& expr);

// initializer list (c++ 2011):

#ifdef _RHEOLEF_HAVE_STD_INITIALIZER_LIST
    vec (const std::initializer_list<vec_concat_value<T,M> >& init_list);
    vec<T,M>& operator= (const std::initializer_list<vec_concat_value<T,M> >& init_
#endif // _RHEOLEF_HAVE_STD_INITIALIZER_LIST
};
```

5.20 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 9.1 [iorheo ialgorithm], page 159).

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&);

// in TMPDIR environment variable or "/tmp" by default
std::string get_tmpdir();
```

6 Forms

6.1 2D – rate of deformation tensor

(Source file: 'nfem/pform_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.

$$b(\mathbf{u}, \tau) = \int_\Omega 2D(\mathbf{u}) : \tau \, dx, \forall u \in V \text{ and } q \in T.$$

where

$$2D(\mathbf{u}) = \nabla \mathbf{u} + \nabla \mathbf{u}^T$$

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");
```

6.2 2D_D – -div 2D operator

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

Synopsis

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

Description

Assembly the form associated to the **div**$(2D(.))$ components of the operator on the finite element space V:

$$a(\mathbf{u},\mathbf{v}) = \int_\Omega 2D(\mathbf{u}) : D(\mathbf{v})\, dx$$

where

$$D_{ij}(u) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

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.

6.3 `curl` – curl operator

(Source file: '`nfem/pform_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:

$$b(\mathbf{u},\mathbf{q}) = \int_\Omega \mathbf{curl}\,\mathbf{u}.\mathbf{q}\, dx$$

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 5.5 [form class], page 43 and Section 5.7 [space class], page 53.

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");
```

6.4 `div` – divergence operator

(Source file: '`nfem/pform_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`:

$$b(\mathbf{u}, q) = \int_\Omega \operatorname{div} \mathbf{u}\, q\, dx$$

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 5.5 [form class], page 43 and Section 5.7 [space class], page 53.

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");
```

6.5 `div_div` – -grad div operator

(Source file: '`nfem/pform_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`:

$$a(\mathbf{u},\mathbf{v}) = \int_\Omega \mathrm{div}(\mathbf{u})\mathrm{div}(\mathbf{v})\,dx$$

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");
```

6.6 `grad` – gradient operator

(Source file: '`nfem/pform_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`:

$$b(u,\mathbf{q}) = \int_\Omega \nabla u.\mathbf{q}\,dx$$

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 5.5 [form class], page 43 and Section 5.7 [space class], page 53.

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");
```

6.7 grad_grad – -Laplacian operator

(Source file: 'nfem/pform_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 5.5 [form class], page 43 and Section 5.7 [space class], page 53.

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");
```

6.8 inv_mass – invert of L2 scalar product

(Source file: 'nfem/pform_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:

$$m(u,v) = \int_\Omega uv \, dx$$

The V space may be either a P0 or P1d discontinuous finite element spaces see Section 5.5 [form class], page 43.

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");
```

6.9 `lumped_mass` – lumped L2 scalar product

(Source file: '`nfem/pform_element/lumped_mass.h`')

Synopsis

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

Example

The use of lumped mass form write:

```
form m(V, "lumped_mass");
```

or (see also Section 6.10 [mass form], page 98):

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

The lumped procedure sums all extra-diagonal terms on the diagonal: let us denote M the original mass matrix, then, the lumped mass matrix is the diagonal matrix defined by:

```
                n
               ----
               \
M1(i,i) =      /    M(i,j)
               ----
                j=1
```

6.10 `mass` – L2 scalar product

(Source file: '`nfem/pform_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 5.5 [form class], page 43.

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);
```

6.11 `s_curl` – curl-like operator for the Stokes stream function computation

(Source file: '`nfem/pform_element/s_curl.h`')

Synopsis

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

Description

Assembly the form associated to the s_curl operator on a finite element space `V`: $$ b(\xi,u) = \int_\Omega u. s\_curl \xi \, dx $$ The M and V space may be a either `P1` or `P2` finite element space. The M space is scalar-valued while the V is vector-valued. See also Section 5.5 [form class], page 43 and Section 5.7 [space class], page 53.

For cartesian coordinate systems, this form coincide with the usual "curl" one (see Section 6.3 [curl form], page 94). In the axisymmetric case:

```
                /
                | (d xi      d xi   )
b(xi,u) =       | (---- ur - ---- uz) r dr dz
                | (d z       d r    )
                / Omega
```

The b form is denoted as "s_curl", for Stokes stream function computation (see Section 6.12 [s_grad_grad form], page 101) as it is closely related to the "curl" operator (see Section 6.3 [curl form], page 94), but differs by the r and 1/r factors, as:

```
                  (       d (r xi)     d xi )
curl(xi) =        ( (1/r) -------- ; - -----)
                  (         d r        d z  )
```

while

```
                  ( d xi       d xi )
s_curl(xi) =      ( ----  ;  - ---- )
                  ( d r        d z  )
```

Notice also that the differentiation is performed on the xi variable here: b(xi,u)=(s_curl(xi),u) while the "curl" form brings the differentiation on the u vector-valued variable: (curl(u),xi), i.e. a transpose formulation.

Orientation and sign fix

The (r,theta,z) coordinate system has positive orientation, thus (z,r,theta) and (z,r) are positive also. But (r,z,theta) and (r,z) are negative : the sign of s_curl is then inverted to obtain the same result as if (z,r) was used.

Example

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

```
geo g("square");
space M(g, "P1");
space V(g, "P1", "vector");
form a(M, V, "s_curl");
```

6.12 s_grad_grad – grad_grad-like operator for the Stokes stream function computation

(Source file: '`nfem/pform_element/s_grad_grad.h`')

Synopsis

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

Description

Assembly the form associated to the -div(grad) variant operator on a finite element space `V`. The V space may be a either `P1` or `P2` finite element space. See also Section 5.5 [form class], page 43 and Section 5.7 [space class], page 53. On cartesian coordinate systems, the form coincide with the "grad_grad" one (see Section 6.7 [grad_grad form], page 97): $$ a(u,v) = \int_\Omega \nabla u . \nabla v \, dx $$ The stream function on tri-dimensionnal cartesian coordinate systems is such that

```
u = curl psi
div psi = 0
```

where u is the velocity field. Taking the curl of the first relation, using the identity:

```
curl(curl(psi)) = -div(grad(psi)) + grad(div(psi))
```

and using the div(psi)=0 relation leads to:

```
-div(grad(psi)) = curl(u)
```

This relation leads to a variational formulation involving the the "grad_grad" and the "curl" forms (see Section 6.7 [grad_grad form], page 97, Section 6.3 [curl form], page 94).

In the axisymmetric case, the stream function psi is scalar ans is defined from the velocity field u=(ur,uz) by (see Batchelor, 6th ed., 1967, p 543):

$$uz = (1/r)\,\frac{d\,psi}{d\,r} \quad \text{and} \quad ur = -\,(1/r)\,\frac{d\,psi}{d\,r}$$

See also http://en.wikipedia.org/wiki/Stokes_stream_function . Multiplying by rot(xi)=(d xi/dr, -d xi/dz), and integrating with r dr dz, we get a well-posed variationnal problem:

```
a(psi,xi) = b(xi,u)
```

with

$$a(psi,xi) = \int_{Omega} \left(\frac{d\,psi}{d\,r}\frac{d\,xi}{d\,r} + \frac{d\,psi}{d\,z}\frac{d\,xi}{d\,z}\right) dr\,dz$$

and

$$b(xi,u) = \int_{Omega} \left(\frac{d\,xi}{d\,z}\,ur - \frac{d\,xi}{d\,r}\,uz\right) r\,dr\,dz$$

Notice that a is symmetric definite positive, but without the 'r' weight as is is usual for axisymmetric standard forms. The b form is named "s_curl", for the Stokes curl variant of the "curl" operator (see Section 6.11 [s_curl form], page 100) as it is closely related to the "curl" operator, but differs by the r and 1/r factors, as:

$$curl(xi) = \left((1/r)\,\frac{d\,(r\,xi)}{d\,r}\,;\, -\frac{d\,xi}{d\,z}\right)$$

while

$$s\_curl(xi) = \left(\frac{d\,xi}{d\,r}\,;\, -\frac{d\,xi}{d\,z}\right)$$

Example

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

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

7 Algorithms

7.1 adapt - mesh adaptation

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

Synopsys

geo adapt (const field& phi); geo adapt (const field& phi, const adapt_option_type& opts);

Description

The function adapt implements the mesh adaptation procedure, based on the gmsh (isotropic) or bamg (anisotropic) mesh generators. The bamg mesh generator is the default in two dimension. For dimension one or three, gmsh is the only generator supported yet. In the two dimensional case, the gmsh correspond to the opts.generator="gmsh".

The strategy based on a metric determined from the Hessian of a scalar governing field, denoted as phi, and that is supplied by the user. Let us denote by H=Hessian(phi) the Hessian tensor of the field phi. Then, |H| denote the tensor that has the same eigenvector as H, but with absolute value of its eigenvalues:

```
|H| = Q*diag(|lambda_i|)*Qt
```

The metric M is determined from |H|. Recall that an isotropic metric is such that M(x)=hloc(x)^(-2)*Id where hloc(x) is the element size field and Id is the identity d*d matrix, and d=1,2,3 is the physical space dimension.

Gmsh isotropic metric

$$M(x) = \frac{\max_{i=0..d-1}(|\lambda_i(x)|)\,Id}{err*hcoef^2*(sup_y(phi(y))-inf_y(phi(y)))}$$

Notice that the denominator involves a global (absolute) normalization sup_y(phi(y))-inf_y(phi(y)) of the governing field phi and the two parameters opts.err, the target error, and opts.hcoef, a secondary normalization parameter (defaults to 1).

Bamg anisotropic metric

There are two approach for the normalization of the metric. The first one involves a global (absolute) normalization:

$$M(x) = \frac{|H(x)|}{err*hcoef^2*(sup_y(phi(y))-inf_y(phi(y)))}$$

The first one involves a local (relative) normalization:

```
                          |H(x))|
  M(x) = ----------------------------------------
         err*hcoef^2*(|phi(x)|, cutoff*max_y|phi(y)|)
```

Notice that the denominator involves a local value `phi(x)`. The parameter is provided by the optional variable `opts.cutoff`; its default value is `1e-7`. The default strategy is the local normalization. The global normalization can be enforced by setting `opts.additional="-AbsError"`.

When choosing global or local normalization ?

When the governing field `phi` is bounded, i.e. when `err*hcoef^2*(sup_y(phi(y))-inf_y(phi(y)))` will converge versus mesh refinement to a bounded value, the global normalization defines a metric that is mesh-independent and thus the adaptation loop will converge.

Otherwise, when `phi` presents singularities, with unbounded values (such as corner singularity, i.e. presents peacks when represented in elevation view), then the mesh adaptation procedure is more difficult. The global normalization divides by quantities that can be very large and the mesh adaptation can diverges when focusing on the singularities. In that case, the local normalization is preferable. Moreover, the focus on singularities can also be controled by setting `opts.hmin` not too small.

The local normalization has been choosen as the default since it is more robust. When your field `phi` does not present singularities, then you can swith to the global numbering that leads to a best equirepartition of the error over the domain.

Implementation

```
struct adapt_option_type {
    typedef std::vector<int>::size_type size_type;
    std::string generator;
    bool isotropic;
    Float err;
    Float errg;
    Float hcoef;
    Float hmin;
    Float hmax;
    Float ratio;
    Float cutoff;
    size_type n_vertices_max;
    size_type n_smooth_metric;
    bool splitpbedge;
    Float thetaquad;
    Float anisomax;
    bool clean;
    std::string additional;
    bool double_precision;
    Float anglecorner;  // angle below which bamg considers 2 consecutive edge to b
                        // the same spline
    adapt_option_type() :
        generator(""),
```

```
        isotropic(true), err(1e-2), errg(1e-1), hcoef(1), hmin(0.0001), hmax(0.3),
        n_vertices_max(50000), n_smooth_metric(1),
        splitpbedge(false), thetaquad(std::numeric_limits<Float>::max()),
        anisomax(1e6), clean(false), additional("-RelError"), double_precision(fals
        anglecorner(0)
    {}
};
template <class T, class M>
geo_basic<T,M>
adapt (
  const field_basic<T,M>& phi,
  const adapt_option_type& options = adapt_option_type());
```

7.2 damped_newton – damped Newton nonlinear algorithm

(Source file: 'nfem/plib/damped-newton.h')

Description

Nonlinear damped Newton algorithm for the resolution of the following problem:

```
F(u) = 0
```

A simple call to the algorithm writes:

```
my_problem P;
field uh (Vh);
damped_newton (P, uh, tol, max_iter);
```

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, odiststream* p_d
  return damped_newton(P, newton_identity_preconditioner(), u, tol, max_iter, p_der
}
```

7.3 `integrate` - integrate a function

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

Description

Integrate a function over a domain by using a quadrature formulae.

Synopsys

template <class T, class M, class Function> T integrate (const geo_basic<T,M>& g, Function f, quadrature_option_type qopt)

Example

```
Float f(const point& x);
...
Float value = integrate (omega, f, quadrature_option);
```

An optional argument specifies the quadrature formulae used for the computation of the integral.

Implementation

```
template <class T, class M, class Function>
T integrate (const geo_basic<T,M>& g, Function f, const quadrature_option_type& qop
{
    space_basic<T,M> Xh (g, "P0");
    field_basic<T,M> int_K_f_dx = riesz (Xh, f, qopt);
    return dot (1, int_K_f_dx);
}
```

7.4 `interpolate` - Lagrange interpolation of a function

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

Description

The function `interpolation` implements the Lagrange interpolation of a function or a class-function.

Synopsys

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

Example

The following code compute the Lagrange interpolation `pi_h_u` of u(x).

```
Float u(const point& x);
...
geo omega("square");
space Xh (omega, "P1");
field pi_h_u = interpolate (Xh, u);
```

Advanced example

It is possible the replace the function `u` by a variable of the `field` type that represents a picewise polynomial function: this invocation allows the reinterpolation of a field on another mesh or with another approximation.

```
geo omega2 ("square2");
space X2h (omega2, "P1");
field uh2 = interpolate (X2h, pi_h_u);
```

Implementation

```
template <class T, class M, class Function>
inline
field_basic<T,M>
interpolate (const space_basic<T,M>& V, Function f)
```

7.5 `level_set` - compute a level set from a function

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

Synopsys

geo level_set (const field& fh);

Description

Given a function `fh` defined in a domain `Lambda`, compute the level set defined by {x in Lambda, fh(x) = 0}. This level set is represented by the `geo` class.

Options

The option class `leve_set_option_type` controls the slit of quadrilaterals into triangles for tridimensional intersected surface and also the zero machine precision, `epsilon`.

Implementation

```
struct level_set_option_type {
    bool split_to_triangle;
    Float epsilon;
    level_set_option_type()
```

```
      : split_to_triangle(true),
        epsilon(100*std::numeric_limits<Float>::epsilon())
        {}
};
template <class T, class M>
geo_basic<T,M> level_set (
  const field_basic<T,M>& fh,
  const level_set_option_type& opt = level_set_option_type());
```

7.6 newton – Newton nonlinear algorithm

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

Description

Nonlinear Newton algorithm for the resolution of the following problem:

```
F(u) = 0
```

A simple call to the algorithm writes:

```
my_problem P;
field uh (Vh);
newton (P, uh, tol, max_iter);
```

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

```
class my_problem {
public:
  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, odiststream *p_derr
    if (p_derr) *p_derr << "# 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_derr) *p_derr << 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;
      }
  }
```

7.7 `riesz` - integrate a function by using quadrature formulae

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

Description

The function `riesz` implements the approximation of an integral by using quadrature formulae.

Synopsys

template <class Function> field riesz (const space& Xh, const Function& f, quadrature_option_type qopt = default_value);

template <class Function> field riesz (const space& Xh, const Function& f, const geo& domain, quadrature_option_type qopt = default_value);

Example

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

```
Float f(const point& x);
...
space Xh (omega_h, "P1");
field lh = riesz (Xh, f);
Float int_f = dot(lh, 1);
```

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

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

where phi_i is the i-th basis function in Xh 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 Xh. Alternative quadrature formulae and order is available by passing an optional variable to riesz.

Options

An optional argument specifies the quadrature formulae used for the computation of the integral. The domain of integration is by default the mesh associated to the finite element space. An alternative domain `dom`, e.g. a part of the boundary can be supplied as an extra argument. This domain can be also a `band` associated to the banded level set method.

Implementation

```
template <class T, class M, class Function>
field_basic<T,M>
riesz (
    const space_basic<T,M>&       Xh,
    const Function&               f,
    const quadrature_option_type& qopt
       = quadrature_option_type(quadrature_option_type::max_family,0))
```

Implementation

```
template <class T, class M, class Function>
field_basic<T,M>
riesz (
    const space_basic<T,M>&       Xh,
    const Function&               f,
    const geo_basic<T,M>&         dom,
    const quadrature_option_type& qopt
       = quadrature_option_type(quadrature_option_type::max_family,0))
```

Implementation

```
template <class T, class M, class Function>
field_basic<T,M>
riesz (
    const space_basic<T,M>&       Xh,
    const Function&               f,
    std::string                   dom_name,
    const quadrature_option_type& qopt
       = quadrature_option_type(quadrature_option_type::max_family,0))
```

Implementation

```
template <class T, class M, class Function>
field_basic<T,M>
riesz (
    const space_basic<T,M>&       Xh,
    const Function&               f,
    const band_basic<T,M>&        gh,
    const quadrature_option_type& qopt
       = quadrature_option_type(quadrature_option_type::max_family,0))
```

7.8 `diag` - get diagonal part of a matrix

(Source file: '`skit/plib2/diag.h`')

Description

This function get the diagonal part of a matrix.

```
csr<Float> a;
dia<Float> d = diag(a);
```

Todo

Build a csr matrix from a diagonal one or from a vector:

```
dia<Float> d;
csr<Float> a = diag(d);
vec<Float> u;
csr<Float> b = diag(u);
```

Implementation

```
template<class T, class M>
dia<T,M> diag (const csr<T,M>& a)
```

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

(Source file: 'skit/plib2/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,
  odiststream *p_derr = 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,
  odiststream *p_derr = 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.

7.10 pcg – conjugate gradient algorithm.

(Source file: '`skit/plib2/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, odiststream *p_derr=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, &derr);
```

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 Vector2, class Preconditioner, class Re
int pcg(const Matrix &A, Vector &x, const Vector2 &Mb, const Preconditioner &M,
        Size &max_iter, Real &tol, odiststream *p_derr = 0, std::string label = "cg
{
    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_derr) (*p_derr) << "[" << 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_derr) (*p_derr) << "[" << 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;
    }
```

7.11 pgmres – generalized minimum residual method

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

Synopsis

```
template <class Matrix, class Vector, class Preconditioner,
  class SmallMatrix, class SmallVector, class Real, class Size>
int pgmres (const Matrix &A, Vector &x, const Vector &b, const Preconditioner &M
  SmallMatrix &H, const SmallVector& dummy,
  Size m, Size &max_iter, Real &tol);
```

Example

The simplest call to pgmres has the folling form:

```
double tol = 1e-7;
size_t max_iter = 100;
size_t m = 6;
boost::numeric::ublas::matrix<double> H(m+1,m+1);
vec<double,sequential> dummy;
int status = pgmres (a, x, b, ic0(a), H, dummy, m, max_iter, tol);
```

Description

pgmres 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 pgmres iterations, m specifies the number of iterations for each restart.

pgmres 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 `pgmres` iterations. Within `pgmres`, 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, `pgmres` 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.

`pgmres` is an iterative template routine.

`pgmres` 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 SmallMatrix, class Vector, class SmallVector, class Size>
void
Update(Vector &x, Size k, SmallMatrix &h, SmallVector &s, Vector v[])
{
  SmallVector y = s;
  // Back solve:
  for (int i = k; i >= 0; i--) {
    y(i) /= h(i,i);
    for (int j = i - 1; j >= 0; j--)
      y(j) -= h(j,i) * y(i);
  }
  for (Size j = 0; j <= k; j++) {
    x += v[j] * y(j);
  }
}
template<class Real>
void 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;
}
template <class Matrix, class Vector, class Preconditioner,
          class SmallMatrix, class SmallVector, class Real, class Size>
int
pgmres (const Matrix &A, Vector &x, const Vector &b, const Preconditioner &M,
      SmallMatrix &H, const SmallVector&, const Size &m, Size &max_iter, Real &tol)
{
  odiststream* p_derr = &derr;
  std::string label = "pgmres";
  if (p_derr) (*p_derr) << "[" << label << "] #iteration residue" << std::endl;
  Vector w;
  SmallVector s(m+1), cs(m+1), sn(m+1);
  Size i;
  Size j = 1;
  Size k;
  Real resid;
  Real normb = norm(M.solve(b));
  Vector r = M.solve(b - A * x);
  Real beta = norm(r);
  if (normb == Real(0)) {
    normb = 1;
  }
  resid = norm(r) / normb;
  if (resid  <= 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));
      resid = abs(s(i+1)) / normb;
      if (p_derr) (*p_derr) << "[" << label << "] " << j << " " << resid << std::en
      if (resid  < 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);
    resid = beta / normb;
    if (p_derr) (*p_derr) << "[" << label << "] " << j << " " << resid << std::endl
    if (resid < tol) {
      tol = resid;
      max_iter = j;
      delete [] v;
      return 0;
    }
  }
  tol = resid;
  delete [] v;
  return 1;
}
```

7.12 pminres – conjugate gradient algorithm.

(Source file: 'skit/plib2/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, odiststream *p_derr=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, &derr);
```

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, odiststream *p_derr = 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_derr) (*p_derr) << "[" << label << "] #iteration residue" << std::endl;
  if (p_derr) (*p_derr) << "[" << label << "] 0 " << norm_r/norm_b << std::endl;
  if (beta2 < 0 || norm_r <= tol*norm_b) {
    tol = norm_r/norm_b;
    max_iter = 0;
    dis_warning_macro ("beta2 = " << beta2 << " < 0: stop");
    return 0;
  }
```

```
Real beta = sqrt(beta2);
Real eta = beta;
Vector Mv = Mr/beta;
Vector  u =  z/beta;
Real c_old = 1.;
Real s_old = 0.;
Real c = 1.;
Real s = 0.;
Vector u_old  (x.ownership(), 0.);
Vector Mv_old (x.ownership(), 0.);
Vector w      (x.ownership(), 0.);
Vector w_old  (x.ownership(), 0.);
Vector w_old2 (x.ownership(), 0.);
for (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) {
      dis_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_derr) (*p_derr) << "[" << 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;
  }
```

7.13 puzawa – Uzawa algorithm.

(Source file: 'skit/plib2/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, odiststream *p_derr=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, &derr);
```

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,
    odiststream *p_derr, 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_derr) (*p_derr) << "[" << 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_derr) (*p_derr) << "[" << 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;
}
```

7.14 `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 5.4 [field class], page 37):

```
cin  >> catchmark("f")   >> fh;
cout << catchmark("u")   << uh
     << catchmark("w")   << wh
     << catchmark("psi") << psih;
```

Assuming its value for output is `"u"`, the corresponding labels will be `"#u0"`, `"#u1"`, `"#u2"`, ...

Implementation

```
class catchmark {
    public:
        catchmark(const std::string& x);
```

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

8 Internal classes

8.1 `basis` - polynomial basis

(Source file: '`nfem/plib/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 8.15 [reference_element iclass], page 142). 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 8.21 [smart_pointer iclass], page 152) class on a `basis_rep` class that is a pure virtual base class for effective bases, e.g. basis_P1, basis_P1, etc.

Implementation

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

// typedefs:

    typedef basis_rep<T>                            rep;
    typedef smart_pointer<rep>                      base;
    typedef typename basis_rep<T>::size_type        size_type;

// allocators:

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

// accessors:

    std::string name() const;
    size_type   degree() const;
    size_type   size (reference_element hat_K) const;

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

    void eval(
```

```
        reference_element    hat_K,
        const point_basic<T>&         hat_x,
        std::vector<T>&  values) const;

    void grad_eval(
        reference_element    hat_K,
        const point_basic<T>&         hat_x,
        std::vector<point_basic<T> >&  values) const;
};
typedef basis_basic<Float> basis;
```

8.2 domain_indirect - a named part of a finite element mesh

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

Description

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

Implementation note

The domain class is splitted into two parts. The first one is the domain_indirect class, that contains the main renumbering features: it acts as a indirect on a geo class(see Section 5.6 [geo class], page 46). The second one is the domain class, that simply contains two smart_pointers: one on a domain_indirect and the second on the geo where renumbering is acting. Thus, the domain class develops a complete geo-like interface, via the geo_abstract_rep pure virtual class derivation, and can be used by the space class (see Section 5.7 [space class], page 53). The split between domain_indirect and domain is necessary, because the geo class contains a list of domain_indirect. It cannot contains a list of domain classes, that refers to the geo class itself: a loop in reference counting leads to a blocking situation in the automatic deallocation.

Implementation

```
template <>
class domain_indirect_basic<sequential> : public smart_pointer<domain_indirect_rep<s
public:

// typedefs:

    typedef domain_indirect_rep<sequential> rep;
    typedef smart_pointer<rep>        base;
    typedef rep::size_type            size_type;
    typedef rep::iterator_ioige       iterator_ioige;
    typedef rep::const_iterator_ioige const_iterator_ioige;

// allocators:
```

```
    domain_indirect_basic ();

    template <class T>
    domain_indirect_basic (
        const geo_basic<T,sequential>&  omega,
        const std::string&              name,
        size_type                       map_dim,
        const communicator&             comm,
        const std::vector<size_type>&   ie_list);

    template <class U>
    domain_indirect_basic (
        array<geo_element_auto<heap_allocator<size_type> >,sequential, heap_allocat
                                        d_tmp,
        const geo_basic<U, sequential>& omega,
        std::vector<index_set>*         ball);

    void resize (size_type n);

// accessors:

    size_type size()     const;
    size_type dis_size() const;
    const distributor& ownership() const;

    const_iterator_ioige ioige_begin() const;
    const_iterator_ioige ioige_end()   const;
          iterator_ioige ioige_begin();
          iterator_ioige ioige_end();

    const geo_element_indirect& oige (size_type ioige) const;

    void set_name (std::string name);
    void set_map_dimension (size_type map_dim);
    std::string name ()    const;
    size_type map_dimension () const;

// i/o:

    odiststream& put (odiststream&) const;
    template <class T>
    idiststream& get (idiststream& ips, const geo_rep<T,sequential>& omega, std::ve
};
```

Implementation

```
template <>
class domain_indirect_basic<distributed> : public smart_pointer<domain_indirect_rep
```

```
public:

// typedefs:

    typedef domain_indirect_rep<distributed>  rep;
    typedef smart_pointer<rep>            base;
    typedef rep::size_type                size_type;

// allocators:

    domain_indirect_basic ();
    template<class T>
    domain_indirect_basic (
        const geo_basic<T,distributed>& omega,
        const std::string&            name,
        size_type                     map_dim,
        const communicator&           comm,
        const std::vector<size_type>& ie_list);

// accessors/modifiers:

    size_type size()     const;
    size_type dis_size() const;
    const distributor& ownership() const;

    const geo_element_indirect& oige (size_type ioige) const;

    void set_name (std::string name);
    void set_map_dimension (size_type map_dim);
    std::string name () const;
    size_type map_dimension () const;

// distributed specific acessors:

    const distributor& ini_ownership() const;
    size_type ioige2ini_dis_ioige (size_type ioige) const;
    size_type ini_ioige2dis_ioige (size_type ini_ioige) const;

// i/o:

    template <class T>
    idistream& get (idistream& ips, const geo_rep<T,distributed>& omega);
    template <class T>
    odistream& put (odistream& ops, const geo_rep<T,distributed>& omega) const;
};
```

8.3 `form_element` - bilinear form on a single element

(Source file: '`nfem/plib/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 8.8 [geo_element iclass], page 130).

Implementation note

The `form_element` class is managed by (see Section 8.21 [smart_pointer iclass], page 152). 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

```
template <class T, class M>
class form_element : public smart_pointer<form_element_rep<T,M> > {
public:

// typedefs:

    typedef form_element_rep<T,M>         rep;
    typedef smart_pointer<rep>            base;
    typedef typename rep::size_type       size_type;
    typedef typename rep::vertex_type     vertex_type;
    typedef typename rep::space_type      space_type;
    typedef typename rep::geo_type        geo_type;
    typedef typename rep::coordinate_type coordinate_type;

// constructors:

    form_element ();
    form_element (
        std::string           name,
        const space_type&     X,
        const space_type&     Y,
        const geo_type&       omega,
        const quadrature_option_type& qopt);

// accessors & modifier:

    void operator() (const geo_element& K, ublas::matrix<T>& m) const;
    virtual bool is_symmetric () const;

// for scalar-weighted forms:

    void set_weight (const field_basic<T,M>& wh) const;
```

```
        bool is_weighted() const;
        const field_basic<T,M>& get_weight () const;

    // for banded level set method:

        bool is_on_band() const;
        const band_basic<T,M>& get_band() const;
        void set_band (const band_basic<T,M>& bh) const;

    };
```

8.4 geo_domain - a named part of a finite element mesh that behaves as a mesh

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

Description

The geo_domain class defines a container for a part of a finite element mesh. This class re-describes the vertices, edges or faces in a compact way, i.e. by skipping unused elements from the surrounding mesh.

Implementation note

The geo_domain class conserves the link to the original mesh such that fields defined on a geo_domain can inter-operate with fields defined on the surrounding mesh.

8.5 geo_domain_indirect_rep - a named part of a finite element mesh

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

Description

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

Implementation note

The geo_domain_indirect_rep class is splitted into two parts. The first one is the domain_indirect class, that contains the main renumbering features: it acts as an indirection on a geo class(see Section 5.6 [geo class], page 46). The second one is the geo class itself, named here the background geo. Thus, the geo_domain_indirect class develops a complete geo-like interface, via the geo_abstract_rep pure virtual class derivation, and can be used by the space class (see Section 5.7 [space class], page 53).

The split between `domain_indirect` and `geo_domain_indirect` is necessary, because the `geo` class contains a list of domain_indirect. The `geo` class cannot contains a list of `geo_domain_indirect` classes, that refers to the `geo` class itself: a loop in reference counting leads to a blocking situation in the automatic deallocation.

8.6 `numbering` - global degree of freedom numbering

(Source file: '`nfem/plib/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

```
template <class T, class M = rheo_default_memory_model>
class numbering : public smart_pointer<numbering_rep<T,M> > {
public:

// typedefs:

  typedef numbering_rep<T,M> rep;
  typedef smart_pointer<rep> base;
  typedef size_t             size_type;

// allocators:

  numbering (std::string name = "");
  numbering (numbering_rep<T,M>* ptr);
  ~numbering() {}

// accessors & modifiers:

  bool is_initialized() const { return base::operator->() != 0; }
  std::string name() const;
  size_type degree () const;
  void set_degree (size_type degree) const;
  bool is_continuous()    const;
  bool is_discontinuous() const { return !is_continuous(); }
  const basis_basic<T>& get_basis() const { return base::data().get_basis(); }

  size_type     ndof   (const geo_size& gs, size_type map_dim) const;
  size_type dis_ndof   (const geo_size& gs, size_type map_dim) const;
  void      dis_idof   (const geo_size& gs, const geo_element& K, std::vector<size_
```

```
  void set_ios_permutations (const class geo_basic<T,M>& omega,
                array<size_type,M>&   idof2ios_dis_idof,
                array<size_type,M>&   ios_idof2dis_idof) const;

// comparator:

  bool operator== (const numbering<T,M>& y) const {
        if (! is_initialized() && ! y.is_initialized()) return true;
        if (! is_initialized() || ! y.is_initialized()) return false;
        return name() == y.name();
  }
// i/o:

  void dump(std::ostream& out = std::cerr) const;
};
```

8.7 edge - Edge reference element

(Source file: ‘nfem/geo_element/edge.icc’)

Description

The edge reference element is K = [0,1].

```
0---------1   x
```

Curved high order Pk edges (k >= 1), in 2d or 3d geometries, are supported. These edges have internal nodes, numbered as:

```
0----2----1     0--2--3--1    0-2-3-4-1
    P2              P3           P4
```

Implementation

```
const size_t dimension = 1;
const size_t measure = 1;
const size_t n_vertex = 2;
const Float vertex [n_vertex][dimension] = {
       { 0 },
       { 1 } };
```

8.8 geo_element - element of a mesh

(Source file: ‘nfem/geo_element/geo_element_v4.h’)

Description

Defines geometrical elements and sides as a set of vertice and edge indexes. This element is obtained after a Piola transformation from a reference element (see Section 8.15 [reference_element iclass], page 142). 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 5.6 [geo class], page 46).

Example

This is the test of geo_element:

```
geo_element_auto<> K;
K.set_name('t') ;
cout << "n_vertices: " << K.size()      << endl
     << "n_edges   : " << K.n_edges()   << endl
     << "dimension : " << K.dimension() << endl << endl;
for(geo_element::size_type i = 0; i < K.size(); i++)
    K[i] = i*10 ;
for(geo_element::size_type i = 0; i < K.n_edges(); i++)
    K.set_edge(i, i*10+5) ;
cout << "vertices: local -> global" << endl;
for (geo_element::size_type vloc = 0; vloc < K.size(); vloc++)
    cout << vloc << "-> " << K[vloc] << endl;
cout << endl
     << "edges: local -> global" << endl;
for (geo_element::size_type eloc = 0; eloc < K.n_edges(); eloc++) {
    geo_element::size_type vloc1 = subgeo_local_vertex(1, eloc, 0);
    geo_element::size_type vloc2 = subgeo_local_vertex(1, eloc, 1);
    cout << eloc << "-> " << K.edge(eloc) << endl
         << "local_vertex_from_edge(" << eloc
         << ") -> (" << vloc1 << ", " << vloc2 << ")" << endl;
}
```

8.9 hack_array - container in distributed environment

(Source file: '`nfem/geo_element/hack_array.h`')

Synopsys

STL-like vector container for a distributed memory machine model. Contrarily to array<T>, here T can have a size only known at compile time. This class is used when T is a geo_element raw class, i.e. T=geo_element_e_raw. The size of the geo_element depends upon the oder and is known only at run-time. For efficiency purpose, the hack_array allocate all geo_elements of the same variant (e.g. edge) and order in a contiguous area, since the coreesponding element size is constant.

Example

A sample usage of the class is:

```
std::pair<size_t,size_t> param (reference_element::t, 3); // triangle, order=3
hack_array<geo_element_raw> x (distributor(100), param);
```

The hack_array<T> interface is similar to those of the array<T> one.

Object requirement

There are many pre-requises for the template objet type T:

```
class T : public T::generic_type {
 typedef variant_type;
 typedef raw_type;
 typedef genetic_type;
 typedef automatic_type;
 static const variant_type _variant;
 static size_t _data_size(const parameter_type& param);
 static size_t _value_size(const parameter_type& param);
};
class T::automatic_type : public T::generic_type {
 automatic_type (const parameter_type& param);
};
class T::generic_type {
 typedef raw_type;
 typedef iterator;
 typedef const_iterator;
 iterator _data_begin();
 const_iterator _data_begin() const;
};
ostream& operator<< (ostream&, const T::generic_type&);
```

Implementation

```
template <class T, class A>
class hack_array<T,sequential,A> : public smart_pointer<hack_array_seq_rep<T,A> > {
public:

// typedefs:

    typedef hack_array_seq_rep<T,A>    rep;
    typedef smart_pointer<rep>            base;

    typedef sequential                    memory_type;
    typedef typename rep::size_type       size_type;
    typedef typename rep::value_type      value_type;
    typedef typename rep::reference       reference;
    typedef typename rep::dis_reference   dis_reference;
    typedef typename rep::iterator        iterator;
    typedef typename rep::const_reference const_reference;
    typedef typename rep::const_iterator  const_iterator;
    typedef typename rep::parameter_type  parameter_type;

// allocators:

    hack_array (const A& alloc = A());
    hack_array (size_type loc_size,          const parameter_type& param, const A&
```

```
    void resize   (const distributor& ownership, const parameter_type& param);
    hack_array (const distributor& ownership, const parameter_type& param, const A&
    void resize   (size_type loc_size,          const parameter_type& param);

// local accessors & modifiers:

    A get_allocator() const             { return base::data().get_allocator(); }
    size_type     size () const         { return base::data().size(); }
    size_type dis_size () const         { return base::data().dis_size(); }
    const distributor& ownership() const { return base::data().ownership(); }
    const communicator& comm() const    { return ownership().comm(); }

    reference       operator[] (size_type i)       { return base::data().operator[]
    const_reference operator[] (size_type i) const { return base::data().operator[]

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

// global modifiers (for compatibility with distributed interface):

    dis_reference dis_entry (size_type dis_i) { return operator[] (dis_i); }
    void dis_entry_assembly()                 {}
    template<class SetOp>
    void dis_entry_assembly(SetOp my_set_op)        {}
    template<class SetOp>
    void dis_entry_assembly_begin (SetOp my_set_op) {}
    template<class SetOp>
    void dis_entry_assembly_end (SetOp my_set_op)   {}

// apply a partition:

#ifdef TODO
    template<class RepSize>
    void repartition (                                // old_numbering for *this
        const RepSize&         partition,             // old_ownership
        hack_array<T,sequential,A>& new_array,        // new_ownership (created)
        RepSize&               old_numbering,         // new_ownership
        RepSize&               new_numbering) const   // old_ownership
        { return base::data().repartition (partition, new_array, old_numbering, new

    template<class RepSize>
    void permutation_apply (                        // old_numbering for *this
        const RepSize&          new_numbering,      // old_ownership
        hack_array<T,sequential,A>&  new_array) const   // new_ownership (already a
        { return base::data().permutation_apply (new_numbering, new_array); }
#endif // TODO
```

```
// i/o:

    odiststream& put_values (odiststream& ops) const { return base::data().put_valu
    idiststream& get_values (idiststream& ips)       { return base::data().get_valu
    template <class GetFunction>
    idiststream& get_values (idiststream& ips, GetFunction get_element)       { ret
    template <class PutFunction>
    odiststream& put_values (odiststream& ops, PutFunction put_element) const { ret
#ifdef TODO
    void dump (std::string name) const { return base::data().dump(name); }
#endif // TODO
};
```

Implementation

```
template <class T, class A>
class hack_array<T,distributed,A> : public smart_pointer<hack_array_mpi_rep<T,A> >
public:

// typedefs:

    typedef hack_array_mpi_rep<T,A>    rep;
    typedef smart_pointer<rep>            base;

    typedef distributed                   memory_type;
    typedef typename rep::size_type       size_type;
    typedef typename rep::value_type      value_type;
    typedef typename rep::reference       reference;
    typedef typename rep::dis_reference   dis_reference;
    typedef typename rep::iterator        iterator;
    typedef typename rep::parameter_type  parameter_type;
    typedef typename rep::const_reference const_reference;
    typedef typename rep::const_iterator  const_iterator;
    typedef typename rep::scatter_map_type scatter_map_type;

// allocators:

    hack_array (const A& alloc = A());
    hack_array (const distributor& ownership, const parameter_type& param, const A&
    void resize   (const distributor& ownership, const parameter_type& param);

// local accessors & modifiers:

    A get_allocator() const             { return base::data().get_allocator(); }
    size_type     size () const         { return base::data().size(); }
    size_type dis_size () const         { return base::data().dis_size(); }
    const distributor& ownership() const { return base::data().ownership(); }
    const communicator& comm() const    { return base::data().comm(); }
```

```
    reference       operator[] (size_type i)       { return base::data().operator[]
    const_reference operator[] (size_type i) const { return base::data().operator[]

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

// global accessor:

    template<class Set, class Map>
    void append_dis_entry (const Set& ext_idx_set, Map& ext_idx_map) const { base::

    template<class Set, class Map>
    void get_dis_entry    (const Set& ext_idx_set, Map& ext_idx_map) const { base::

    template<class Set>
    void append_dis_indexes (const Set& ext_idx_set)  { base::data().append_dis_inde

    template<class Set>
    void set_dis_indexes    (const Set& ext_idx_set)  { base::data().set_dis_indexes

    const_reference dis_at (size_type dis_i) const { return base::data().dis_at (dis

    // get all external pairs (dis_i, values):
    const scatter_map_type& get_dis_map_entries() const { return base::data().get_di

// global modifiers (for compatibility with distributed interface):

    dis_reference dis_entry (size_type dis_i)       { return base::data().dis_entry

    void dis_entry_assembly()                       { return base::data().dis_entry_

    template<class SetOp>
    void dis_entry_assembly       (SetOp my_set_op) { return base::data().dis_entry_
    template<class SetOp>
    void dis_entry_assembly_begin (SetOp my_set_op) { return base::data().dis_entry_
    template<class SetOp>
    void dis_entry_assembly_end   (SetOp my_set_op) { return base::data().dis_entry_

// apply a partition:

    template<class RepSize>
    void repartition (                              // old_numbering for *this
        const RepSize&        partition,            // old_ownership
        hack_array<T,distributed>& new_array,            // new_ownership (created)
        RepSize&              old_numbering,        // new_ownership
        RepSize&              new_numbering) const  // old_ownership
        { return base::data().repartition (partition.data(), new_array.data(), old_n
```

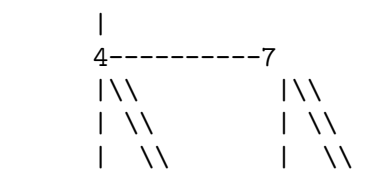
```
#ifdef TODO
    template<class RepSize>
    void permutation_apply (                        // old_numbering for *this
        const RepSize&          new_numbering,      // old_ownership
        hack_array<T,distributed,A>& new_array) const   // new_ownership (already a
        { base::data().permutation_apply (new_numbering.data(), new_array.data()); }

    void reverse_permutation (                                  // old_ownership for
        hack_array<size_type,distributed,A>& inew2dis_iold) const   // new_ownershi
        { base::data().reverse_permutation (inew2dis_iold.data()); }
#endif // TODO

// i/o:

    odiststream& put_values (odiststream& ops) const { return base::data().put_valu
    idiststream& get_values (idiststream& ips)       { return base::data().get_valu
#ifdef TODO
    void dump (std::string name) const      { return base::data().dump(name); }
#endif // TODO

    template <class GetFunction>
    idiststream& get_values (idiststream& ips, GetFunction get_element)
             { return base::data().get_values(ips, get_element); }
    template <class PutFunction>
    odiststream& put_values (odiststream& ops, PutFunction put_element) const
             { return base::data().put_values(ops, put_element); }

    template <class PutFunction, class Permutation>
    odiststream& permuted_put_values (
        odiststream&                        ops,
        const Permutation&                  perm,
        PutFunction                         put_element) const
             { return base::data().permuted_put_values (ops, perm.data(), put_eleme
};
```

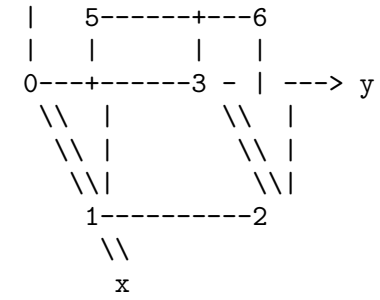
8.10 hexahedron - Hexaedron reference element

(Source file: 'nfem/geo_element/hexahedron.icc')

Description

The hexahedron reference element is [-1,1]^3.

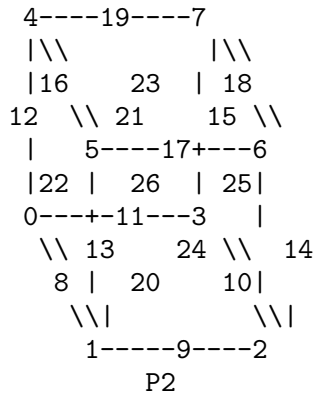




Curved high order Pk hexaedra (k >= 1) in 3d geometries are supported. These hexaedra have additional edge-nodes, face-nodes and internal volume-nodes.

These nodes are numbered as

first vertex, then edge-node, following the edge numbering order and orientation, then face-nodes following the face numbering order and orientation, and finally the face internal nodes, following the hexaedron lattice. See below for edges and faces numbering and orioentation.



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. Notice that the edge-nodes and face-nodes numbering slighly differ from those used in the `gmsh` mesh generator when using high-order elements. This difference is handled by the `msh2geo` mesh file converter (see Section 4.4 [msh2geo command], page 20).

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 } };
```

8.11 `point` - Point reference element

(Source file: '`nfem/geo_element/point.icc`')

Description

The point reference element is defined for convenience. It is a 0-dimensional element with measure equal to 1.

Implementation

```
const size_t dimension = 0;
const size_t measure = 1;
```

8.12 `prism` - Prism reference element

(Source file: '`nfem/geo_element/prism.icc`')

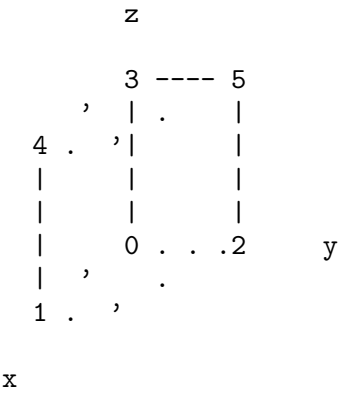
Description

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.



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) },
        { 3, 4, 5, size_t(-1) },
        { 0, 1, 4, 3 },
        { 1, 2, 5, 4 },
        { 0, 3, 5, 2 } };
const size_t  n_edge = 9;
const size_t edge [n_edge][2] = {
        { 0, 1 },
        { 1, 2 },
        { 2, 0 },
        { 0, 3 },
        { 1, 4 },
        { 2, 5 },
        { 3, 4 },
        { 4, 5 },
        { 5, 3 } };
```

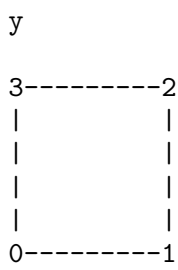
8.13 `quadrangle` - Quadrangular reference element

(Source file: '`nfem/geo_element/quadrangle.icc`')

Description

The quadrangular reference element is [-1,1]^2.

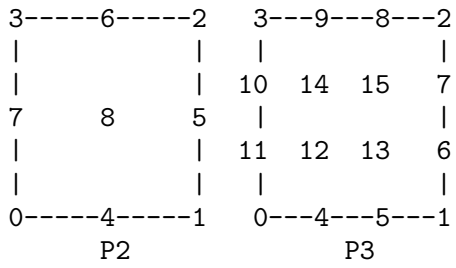
Numbering



Curved high order Pk quadrangles (k >= 1), in 2d or 3d geometries, are supported. These quadrangles have additional edge-nodes and face-nodes.

These nodes are numbered as

first vertex, then edge-node, following the edge numbering order and orientation, and finally the face internal nodes, following the quadrangle lattice. See below for edge numbering and orientation.



Implementation

```
const size_t dimension = 2;
const Float  measure = 4;
const size_t n_vertex = 4;
const 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 quadrature - quadrature formulae on the reference lement

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

Synopsys

The quadrature class defines a container for a quadrature formulae on the reference element (see Section 8.15 [reference_element iclass], page 142). This container stores the nodes coordinates and the weights.

The constructor takes two arguments

the reference element K and the order r of the quadrature formulae. The formulae is exact when computing the integral of a polynom p that degree is less or equal to order r.

$$\int_K p(x)\,dx = \sum_{q=1}^{n} p(x_q)\,w_q$$

Limitations

The formulae is optimal when it uses a minimal number of nodes n. Optimal quadrature formula are hard-coded in this class. Not all reference elements and orders are yet implemented. This class will be completed in the future.

Implementation

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

// typedefs:

    typedef typename quadrature_on_geo<T>::size_type                size_type;
    typedef quadrature_option_type::family_type                     family_type;
    typedef typename std::vector<weighted_point<T> >::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;
    template<class U>
    friend std::ostream& operator<< (std::ostream&, const quadrature<U>&);
protected:
    quadrature_option_type        _options;
    mutable quadrature_on_geo<T>  _quad [reference_element::max_variant];
    mutable std::vector<bool>     _initialized;
    void _initialize (reference_element hat_K) const;
private:
    quadrature (const quadrature<T>&);
    quadrature operator= (const quadrature<T>&);
};
```

8.15 `reference_element` - reference element

(Source file: '`nfem/geo_element/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 variant_type { p, t, q ..., H, ...};
    // in an automatically generated file :

    typedef size_type variant_type;
    static const variant_type
        p = 0,
        e = 1,
        t = 2,
        q = 3,
        T = 4,
        P = 5,
        H = 6,
        max_variant = 7;

// allocators/deallocators:

    reference_element (variant_type x = max_variant)
      : _x(x) { assert_macro (x >= 0 && x <= max_variant, "invalid type " << x); }

// accessors:

    variant_type variant() const { return _x; }
    char name() const { return _name[_x]; }
    size_type dimension() const { return _dimension[_x]; }
    friend Float measure (reference_element hat_K);
    size_type size() const { return _n_vertex[_x]; }
    size_type n_subgeo(size_type subgeo_dim) const { return n_subgeo (variant(), su
    size_type n_edge() const { return n_subgeo(1); }
    size_type n_face() const { return n_subgeo(2); }
    size_type subgeo_size (size_type subgeo_dim, size_type loc_isid) const {
        return subgeo_n_node (_x, 1, subgeo_dim, loc_isid); }
    size_type subgeo_local_vertex(size_type subgeo_dim, size_type loc_isid, size_ty
        return subgeo_local_node (_x, 1, subgeo_dim, loc_isid, loc_jsidvert); }

    void set_variant (variant_type x) { _x = x; }
    void set_variant (size_type n_vertex, size_type dim) { _x = variant (n_vertex,
    void set_name    (char name);

// helpers:

    static variant_type variant  (char name);
    static variant_type variant  (size_type n_vertex, size_type dim);
    static char         name     (variant_type variant) { return _name     [variant]
    static size_type dimension   (variant_type variant) { return _dimension[variant]
    static size_type n_vertex    (variant_type variant) { return _n_vertex [variant]
    static size_type n_node      (variant_type variant, size_type order);

    static size_type n_sub_edge    (variant_type variant);
    static size_type n_sub_face    (variant_type variant);
```

```
    static size_type n_subgeo      (variant_type variant, size_type subgeo_dim);
    static size_type subgeo_n_node (variant_type variant, size_type order, size_type
    static size_type subgeo_local_node (variant_type variant, size_type order, size_

    static variant_type first_variant_by_dimension (size_type dim) {
                return _first_variant_by_dimension[dim]; }
    static variant_type  last_variant_by_dimension (size_type dim) {
                return _first_variant_by_dimension[dim+1]; }

    static size_type first_inod_by_variant (variant_type variant, size_type order,
    static size_type  last_inod_by_variant (variant_type variant, size_type order,
       { return first_inod_by_variant (variant, order, subgeo_variant+1); }
    static size_type first_inod (variant_type variant, size_type order, size_type s
       { return first_inod_by_variant (variant, order, first_variant_by_dimension(s
    static size_type  last_inod (variant_type variant, size_type order, size_type s
       { return first_inod_by_variant (variant, order,  last_variant_by_dimension(s
    static void init_local_nnode_by_variant (size_type order, boost::array<size_type

protected:
// constants:

    static const char      _name [max_variant];
    static const size_type _dimension [max_variant];
    static const size_type _n_vertex [max_variant];
    static const variant_type _first_variant_by_dimension[5];

// data:

    variant_type _x;
};
```

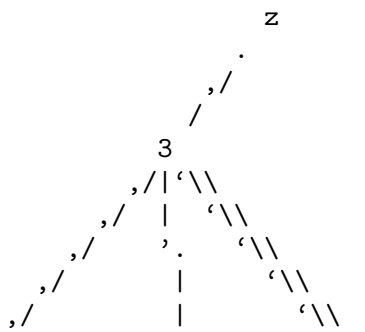
8.16 `tetrahedron` - Tetraedron reference element

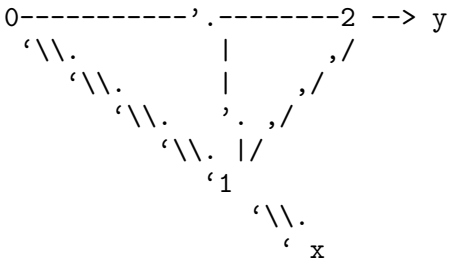
(Source file: '`nfem/geo_element/tetrahedron.icc`')

Description

The tetrahedron reference element is

```
        K = { 0 < x < 1 and 0 < y < 1-x and 0 < z < 1-x-y }
```

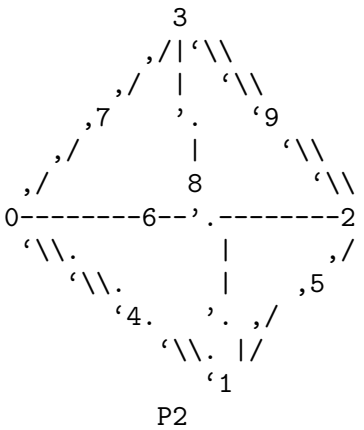




Curved high order Pk tetrahedra (k >= 1) in 3d geometries are supported. These tetrahedra have additional edge-nodes, face-nodes and internal volume-nodes.

These nodes are numbered as

first vertex, then edge-node, following the edge numbering order and orientation, then face-nodes following the face numbering order and orientation, and finally the face internal nodes, following the tetrahedron lattice. See below for edges and faces numbering and orioentation.



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. Notice that the edge-nodes and face-nodes numbering slighly differ from those used in the `gmsh` mesh generator when using high-order elements. This difference is handled by the `msh2geo` mesh file converter (see Section 4.4 [msh2geo command], page 20).

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 } };
```

8.17 `triangle` - Triangle reference element

(Source file: '`nfem/geo_element/triangle.icc`')

Description

The triangle reference element is

```
K = { 0 < x < 1 and 0 < y < 1-x }
```

Numbering

```
y

2
| +
|   +
|     +
|       +
0---------1   x
```

Curved high order Pk triangles (k >= 1), in 2d or 3d geometries, are supported. These triangles have additional edge-nodes and face-nodes.

These nodes are numbered as

first vertex, then edge-node, following the edge numbering order and orientation, and finally the face internal nodes, following the triangle lattice. See below for edge numbering and orientation.

```
2               2               2
| +             | +             | +
|   +           7   6           9   8
5     4         |     +         10 14  7
|       +       8   9   5       11 12 13 6
```

```
|         +     |         +     |         +
0-----3-----1   0---3---4---1   0--3--4--5--1
      P2              P3              P4
```

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 } };
```

8.18 index_set - a set of indexes

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

Synopsys

A class for: l = {1,3,...9} i.e. a wrapper for STL set<size_t> with some assignment operators, such as l1 += l2. This class is suitable for use with the array<T> class, as array<index_set> (see Section 5.11 [array class], page 66).

Implementation

```
class index_set : public std::set<std::size_t> {
public:

// typedefs:

  typedef std::set<std::size_t>    base;
  typedef std::size_t              value_type;
  typedef std::size_t              size_type;

// allocators:

  index_set ();
  index_set (const index_set& x);
  index_set& operator= (const index_set& x);
  template <int N>
  index_set& operator= (size_type x[N]);
  void clear ();
```

```
// basic algebra:

  void       insert     (size_type dis_i);    // a := a union {dis_i}
  index_set& operator+= (size_type dis_i);    // idem
  index_set& operator+= (const index_set& b); // a := a union b

  // a := a union b
  void inplace_union        (const index_set& b);
  void inplace_intersection (const index_set& b);

  // c := a union b
  friend void set_union        (const index_set& a, const index_set& b, index_set&
  friend void set_intersection (const index_set& a, const index_set& b, index_set&

// io:

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

// boost mpi:

  template <class Archive>
  void serialize (Archive& ar, const unsigned int version);
};
```

8.19 `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);
};
```

8.20 heap_allocator - heap-based allocator

(Source file: 'util/lib/heap_allocator.h')

Description

Heap allocators are generally used when there is a lot of allocation and deallocation of small objects. For instance, this is often the case when dealing with `std::list` and `std::map`.

Heap-based allocator is conform to the STL specification of allocators. It does not "free" the memory until the heap is destroyed.

This allocator handles an a priori unlimited area of memory: a sequence of growing chunks are allocated. For a limited memory handler in the same spirit, see "stack_allocator"(9).

Example

```
typedef map <size_t, double, less<size_t>, heap_allocator<pair<size_t,double> >
map_type a;
a.insert (make_pair (0, 3.14));
a.insert (make_pair (1, 1.17));
for (map_type::iterator iter = a.begin(), last = a.end(); iter != last; iter++)
  cout << (*iter).first << " " << (*iter).second << endl;
}
```

Implementation

```
template <typename T>
class heap_allocator {
```

```
protected:
    struct handler_type; // forward declaration:
public:

// typedefs:

    typedef size_t         size_type;
    typedef std::ptrdiff_t difference_type;
    typedef T*             pointer;
    typedef const T*       const_pointer;
    typedef T&             reference;
    typedef const T&       const_reference;
    typedef T              value_type;

// constructors:

    heap_allocator() throw()
      : handler (new handler_type)
    {
    }
    heap_allocator (const heap_allocator& ha) throw()
      : handler (ha.handler)
    {
        ++handler->reference_count;
    }
    template <typename U>
    heap_allocator (const heap_allocator<U>& ha) throw()
      : handler ((typename heap_allocator<T>::handler_type*)(ha.handler))
    {
        ++handler->reference_count;
    }
    ~heap_allocator() throw()
    {
        check_macro (handler != NULL, "unexpected null mem_info");
        if (--handler->reference_count == 0) delete handler;
    }
    // Rebind to allocators of other types
    template <typename U>
    struct rebind {
        typedef heap_allocator<U> other;
    };

// 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);
    }
    // C++ 2011: default construct a value of type T at the location referenced by
    void construct (pointer p) { new ( reinterpret_cast<void*>(p) ) T(); }

    void destroy (pointer p)
    {
        // call destructor directly:
        (p)->~T();
    }

// allocate raw memory

    pointer allocate (size_type n, const void* = NULL)
    {
        return pointer (handler->raw_allocate (n*sizeof(T)));
    }
    void deallocate (pointer p, size_type n)
    {
        // No need to free heap memory
    }
    const handler_type* get_handler() const {
        return handler;
    }

// data:

protected:
    handler_type* handler;
    template <typename U> friend class heap_allocator;
};
```

8.21 smart_pointer, smart_pointer_clone - reference counted safe pointer with true copy semantic

(Source file: 'util/lib/smart_pointer.h')

Description

Here is a convenient way to implement a true copy semantic, by using shallow copies and reference counting, in order to minimise memory copies. This concept is generally related to the *smart pointer* method for managing memory.

The true semantic copy is defined as follows: if an object `A` is assigned to `B`, such as `A = B`, every further modification on `A` or `B` does not modify the other.

Notice that this class differs from the `boost::shared\_ptr` class that implements safe pointers without the true copy semantic.

Clone variant

The `smart_pointer_clone` variant uses a `T* T::clone() const` member function instead of the usual `T::T()` copy constructor for obtaining a true copy of the data. This variant is motivated as follows: when using hierarchies of derived classes (also known as polymorphic classes), the usual copy is not possible because c++ copy constructors cannot be virtual, so you cannot make a copy this way. This is a well-known problem with C++'s implementation of polymorphism.

We uses a solution to the non-virtual copy constructor problem which is suggested by Ellis and Stroustrup in "The Annotated LRM". The solution is to require the `T` class to provide a virtual clone method for every class which makes a copy using new and the correct copy constructor, returning the result as a pointer to the superclass `T`. Each subclass of `T` overloads this function with its own variant which copies its own type. Thus the copy operation is now virtual and furthermore is localised to the individual subclass.

Nocopy variant

This variant of the smart pointer is designed for use on objects that cannot (or must not) be copied. An example would be when managing an object that contains, say, a file handle. It is essential that this not be copied because then you get the problem of deciding which copy is responsible for closing the file. To avoid the problem, wrap the file handle in a class and then manage a unique instance of it using a `smart_pointer_nocopy`. This ensures that the file handle cannot be copied and is closed when the last alias is destroyed.

The interface to the nocopy variant is the same as `smart_pointer` but with all operations that perform copying forbidden. In fact, because all three variants are instances of a common superclass, the forbidden methods do exist but will cause an error and exit if they are called.

The following modifiers cannot be used because they use copying of the pointed-to object and will thereore cause an error:

```
T* operator-> ();
T& operator* ();
T* pointer ();
T& data ();
```

Eferences

[1] A. Geron and F. Tawbi,
Pour mieux developper avec C++ : design pattern, STL, RTTI et smart pointers,
InterEditions, 1999. Page 118.
[2] STLplus, http://stlplus.sourceforge.net/stlplus3/docs/smart_ptr.html
for the clone and nocopy variants.

Implementation

```
template <class T, class C>
class smart_pointer_base {
public:

// allocators:

  smart_pointer_base (T* p = 0);
  smart_pointer_base (const smart_pointer_base<T,C>&);
  smart_pointer_base<T,C>& operator= (const smart_pointer_base<T,C>&);
  ~smart_pointer_base ();

// accessors:

  const T* pointer    () const;
  const T& data       () const;
  const T* operator-> () const;
  const T& operator*  () const;

// modifiers:

  T* pointer    ();
  T& data       ();
  T* operator-> ();
  T& operator*  ();

// implementation:
private:
  struct counter {
    T*  _p;
    int _n;
    counter (T* p = 0);
    ~counter ();
    int operator++ ();
    int operator-- ();
  };
  counter *_count;
#ifndef TO_CLEAN
public:
  int reference_counter() const { return _count != 0 ? _count->_n : -1; }
#endif // TO_CLEAN
```

```
};
```

8.22 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 std::ptrdiff_t difference_type;
    typedef T*             pointer;
    typedef const T*       const_pointer;
    typedef T&             reference;
    typedef const T&       const_reference;
    typedef T              value_type;
```

```
// constructors:

    stack_allocator() throw()
      : handler (new handler_type)
    {
    }
    stack_allocator (unsigned char* stack, size_t stack_size) throw()
      : handler (new handler_type (stack, stack_size))
    {
        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);
    }
    // C++ 2011: default construct a value of type T at the location referenced by p
    void construct (pointer p) { new ( reinterpret_cast<void*>(p) ) T(); }

    void destroy (pointer p)
    {
        // call destructor directly:
        (p)->~T();
    }

// allocate raw memory

    pointer allocate (size_type n, const void* = NULL)
    {
        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();
}
```

9 Internal algorithms

9.1 `iorheo` - input and output functions and manipulation

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

Small pieces of code

input geo in standard file format:

```
cin >> g;
```

output geo in standard file format:

```
cout << g;
```

output geo in gnuplot format:

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

9.2 `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.

Examples

```
typedef map <size_t, double, less<size_t>, heap_allocator<pair<size_t,double> > >
cout << typeid_name_macro (map_type);
```

Implementation

```
extern std::string typeid_name (const char* name, bool do_indent);

/// @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
```

10 Internal others

10.1 `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 LIBS, respectively, If not, set the shell variable rheo_have_ginac to "no".

`RHEO_CHECK_CLN`

Check to see if library `-lcln` exists. If so, set the shell variable `rheo_have_cln` to "yes", defines HAVE_CLN and substitues INCLUDES_CLN and LADD_CLN for adding in CFLAGS and LIBS, respectively, If not, set the shell variable no "no". Includes and libraries path are searched from a given shell variable `rheo_dir_cln`. This shell variable could be set for instance by an appropriate `--with-cln`=*value_dir_cln* option. The default value is `/usr/local/math`.

`RHEO_CHECK_SPOOLES_2_0`

Check to see if spooles library has old version 2.0 since `FrontMtx_factorInpMtx` profile has changed in version 2.2. If so, defines HAVE_SPOOLES_2_0. This macro is called by RHEO_CHECK_SPOOLES.

`RHEO_CHECK_TAUCS`

Check to see if taucs library and headers exists. If so, set the shell variable "rheo_have_taucs" to "yes", defines HAVE_TAUCS and substitues INCLUDES_TAUCS and LADD_TAUCS for adding in CXXFLAGS and LIBS, respectively, If not, set the shell variable to "no". Includes and libraries options are given shell variable $rheo_ldadd_taucs and $rheo_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 LIBS, respectively, If not, set the shell variable to "no". Includes and libraries path are searched from given shell variable $rheo_dir_zlib/lib and $rheo_incdir_zlib. Default value for $rheo_incdir_zlib is $rheo_dir_zlib/include. These shell variables could be set for instance by appropriates "–with-zlib="'dir_zlib' and "–with-zlib-includes="'incdir_zlib' options.

`RHEO_CHECK_SPOOLES`

Check to see if spooles library and headers exists. If so, set the shell variable "rheo_have_spooles" to "yes", defines HAVE_SPOOLES and substitues INCLUDES_SPOOLES and LADD_SPOOLES for adding in CXXFLAGS and LIBS, respectively, If not, set the shell variable to "no". Includes and libraries path are searched from given shell variable "rheo_libdir_spooles" and "rheo_incdir_spooles". These shell variables could be set for instance by appropriates "–with-spooles="'libdir_spooles' and "–with-spooles-includes="'incdir_spooles' options.

`RHEO_CHECK_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 LIBS, respectively, If not, set the shell variable to "no". Includes and libraries path are searched from given shell variable "rheo_libdir_umfpack" and "rheo_incdir_umfpack". These shell variables could be set for instance by appropriates "–with-umfpack="'libdir_umfpack' and "–with-umfpack-includes="'incdir_umfpack' options.

`RHEO_CHECK_MALLOC_DBG`

Check to see if malloc debug library -lmalloc_dbg and corresponding header <malloc_dbg.h> exists. If so, set the shell variable `rheo_have_malloc_dbg` to "yes", defines HAVE_MALLOC_DBG, add `-I`*dir_malloc_dbg*`/include` to CFLAGS, add *dir_malloc_dbg*`/lib/libmalloc_dbg.a` to LIBS. Here, *dir_malloc_dbg* is the directory such that *dir_malloc_dbg*`/bin` appears in PATH and the command *dir_malloc_dbg*`/bin/malloc_dbg` exists. If not, set the variable to "no". Set also LIBS_MALLOC_DBG to these flags.

`RHEO_CHECK_DMALLOC`

Check whether the dmalloc package exists and set the corresponding shell value "rheo_have_dmalloc" and HAVE_DMALLOC (in Makefile.am and config.h) accordingly, create LDADD_DMALLOC and LDADD_DMALLOCXX Makefile.am variables.

`RHEO_CHECK_NAMESPACE`

Check whether the namespace feature is supported by the C++ compiler. value. So, try to compile the following code:

```
namespace computers {
   struct keyboard { int getkey() const { return 0; } };
}
namespace music {
```

```
            struct keyboard { void playNote(int note); };
         }
         namespace music {
            void keyboard::playNote(int note) { }
         }
         using namespace computers;
int main() {
   keyboard x;
   int z = x.getkey();
   music::keyboard y;
   y.playNote(z);
   return 0;
}
```

If it compile, set the corresponding shell variable "rheo_have_namespace" to "yes" and defines HAVE_NAMESPACE. If not, set the variable no "no".

`RHEO_CHECK_STD_NAMESPACE`

Some compilers (e.g. GNU C++ 2.7.2) does not support the full namespace feature. Nevertheless, they support the "std:" namespace for the C++ library. is supported by the C++ compiler. The following code is submitted to the compiler:

```
#include<vector.h>
extern "C" void exit(int);
int main() {
    std::vector<int> x(3);
         return 0;
}
```

If it compile, set the corresponding shell variable "rheo_have_std_namespace" to "yes" and defines HAVE_STD_NAMESPACE. If not, set the variable no "no".

`RHEO_PROG_GNU_MAKE`

Find command make and check whether make is GNU make. If so, set the corresponding shell variable "rheo_prog_gnu_make" to "yes" and 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++ com-

piler 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_INTEL

Check wheter we are using INTEL C++ compiler. If so, set the shell variable "ac_cv_prog_icc" to "yes" and define HAVE_INTEL_CXX. If not, set the shell variable no "no". The shell variable is also exported for sub-shells, such as ltconfig from libtool.

RHEO_RECOGNIZE_CXX

Check wheter we are able to recognize the C++ compiler. Tested compilers:

```
The GNU   C++ compiler that defines: __GNUC__  (egcs-1.1.1)
The INTEL C++ compiler that defines: __ICC     (ICPC-12)
```

If so, 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 icc cl)
AC_PROG_CXX(c++ g++ cxx icpc KCC CC CC cc++ xlC aCC)
RHEO_RECOGNIZE_CXX
```

RHEO_GXX2011_PRE

Check for the "-std=c++0x" support for g++. Requires a recent version of the GNU C++ compiler (>= 4.5).

RHEO_GXX2011

Check for the "-std=c++11" support for g++. Requires a recent version of the GNU C++ compiler (>= 4.7).

`RHEO_OPTIMIZE_CXX`

Set some optimization flags associated to the recognized C++ compiler and platform.

`RHEO_CHECK_LATEX_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", 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 partitioner 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_TRILINOS`

Check for the "trilinos" distributed preconditioner libraries. Defines HAVE_TRILINOS and substitues INCLUDES_TRILINOS and LDADD_TRILINOS. Requires the MPI and SCOTCH libraries.

`RHEO_CHECK_PASTIX`

Check for the "pastix" sequential or distributed direct solver libraries, depending on the "rheo_use_distributed" shell variable. Defines HAVE_PASTIX and substitues INCLUDES_PASTIX and LDADD_PASTIX.

`RHEO_CHECK_MUMPS`

Check for the "mumps" distributed direct solver libraries. Defines HAVE_MUMPS and substitues INCLUDES_MUMPS and LDADD_MUMPS. Requires the MPI and SCOTCH libraries.

`RHEO_CHECK_STD_INITIALIER_LIST`

Some compilers (e.g. GNU C++ 4.4.x) does not support the std::initializer_list feature. Set the corresponding shell variable "rheo_have_std_initializer_list" to "yes" and defines HAVE_STD_INITIALIZER_LIST. If not, set the variable no "no".

`RHEO_CHECK_CGAL`

Check for the "cgal" computational geometry library. Defines HAVE_CGAL and substitues INCLUDES_CGAL and LDADD_CGAL.

`RHEO_DEBIAN_FIX_LIBTOOL`

Fix rpath libtool issue for debian packaging. See also http://wiki.debian.org/RpathIssue

11 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`.

11.1 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
```

11.1.1 In which order does the things build ?

Let us look at with details the configure files flow:

```
[acinclude.m4] -----> aclocal* -----------> [aclocal.m4]

[configure.ac] -+
----------------+---> autoconf* ----------> configure
[aclocal.m4] ---+

[Makefile.am] ------> automake* ----------> Makefile.in

[config.h.in] -+                        +-> config.h
               |                        |
Makefile.in ---+----> configure* -------+-> Makefile
               |                        |
[config.mk.in] +                        +-> config.mk
```

11.1.2 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.

11.2 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
```

11.2.1 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
```

11.2.2 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 update blas3_tst.c
sdiff blas3_tst.c blas3_tst.c.new | more
```

and then edit '`blas3_tst.c`' to integrate the modifications of '`blas3_tst.c.new`', generated by another developer. When it is done, go to the top directory and enter,

```
make status
```

It prints new:

```
Modified                skit/lib/blas1_tst.c
Update                  skit/lib/blas2_tst.c
Modified                skit/lib/blas3_tst.c
```

The situation becomes mature:

```
make update
```

It will update the '`blas2_tst.c`'. Finally,

```
make save
```

that will save modified '`blas1_tst.c`' and '`blas3_tst.c`'.

11.2.3 I have deleted a source file...

I have entered:

```
rm Makefile.am
```

...aie !

How to restaure the file, now ?

Enter:

```
cvs update Makefile.am
```

You restaure the last available version of the missing file in the most up-to-date Rheolef distribution.

Appendix A GNU General Public License

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

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

O

P

Q

R

S

T

U

V

Program Index

Class Index

Form Index

2

C

D

G

I

L

M

S

Approximation Index

B

P

Function Index

File Format Index

Related Tool Index

A

B

C

D

G

H

I

K

M

P

S

T

U

V

Short Contents

Table of Contents