```
#include "hermes2d.h"
#include "definitions.h"

using namespace RefinementSelectors;

int main(int argc, char* argv[])
{
  // Choose a Butcher's table or define your own.
  ButcherTable bt(butcher_table_type);

  // Load the mesh.
  Mesh mesh;
  MeshReaderH2D mloader;
  mloader.load("domain.mesh", &mesh);

  // Perform initial mesh refinements.
  for(int i = 0; i < INIT_REF_NUM; i++) mesh.refine_all_elements();
  mesh.refine_towards_boundary("Boundary_air", INIT_REF_NUM_BDY);
  mesh.refine_towards_boundary("Boundary_ground", INIT_REF_NUM_BDY);

  // Previous and next time level solutions.
  ConstantSolution<double> sln_time_prev(&mesh, TEMP_INIT);
  Solution<double> sln_time_new(&mesh);
```

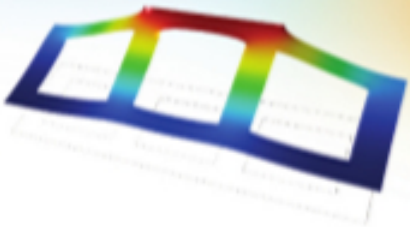
www.hpfem.org/hermes

Hermes2D

Rapid *hp*-FEM & *hp*-DG Solver Toolkit

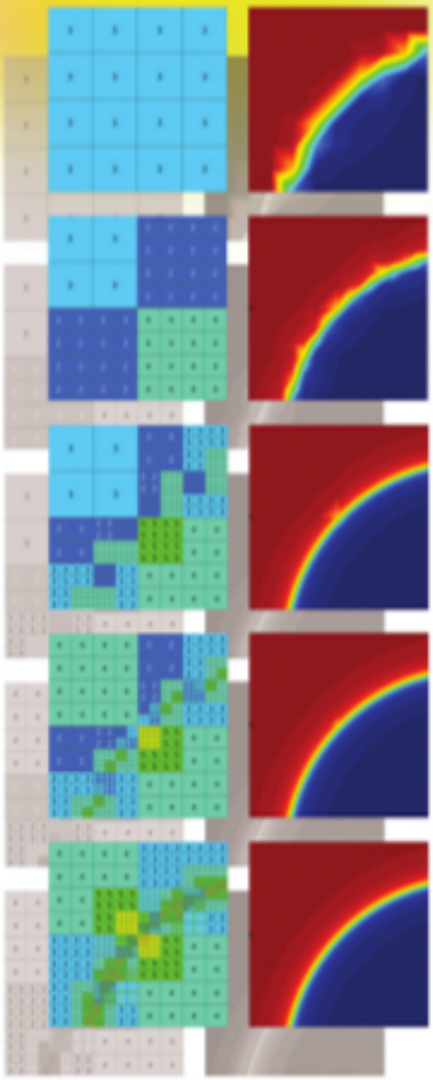
Basic characteristics

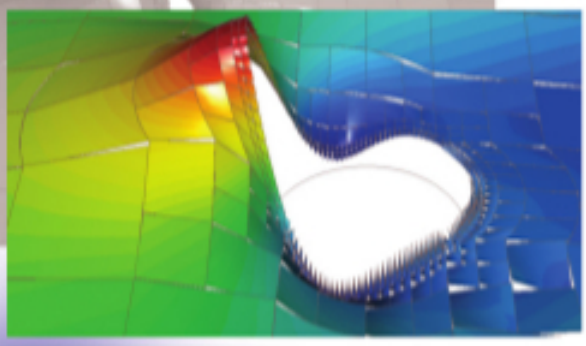
Open-Source C++ library
Linux/Windows
Free software distributed under the GNU GPLv2

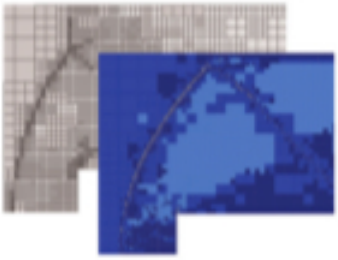


Main features

L2, H1, H-curl, H-div Sobolev spaces
Elements of polynomial degree up to 10
Hanging nodes of arbitrary order
Component-specific meshes with no projections or interpolations
Time-adaptivity with many pre-implemented Runge-Kutta methods
Coupled problems solved with *hp*-DG and *hp*-FEM side by side
hp-adaptivity based on reference solution and local projections







Solvers interfaces

UMFPACK, PARALUTION
PETSc, MUMPS
Trilinos, MatLab
MatrixMarket matrix exports
Exception safe API

Library implementation features

OpenMP parallelization
C++ templates for unified handling of real and complex problems
Own OpenGL visualization & VTK outputs of meshes, spaces, solutions
XML, BSON save/load of the most important classes

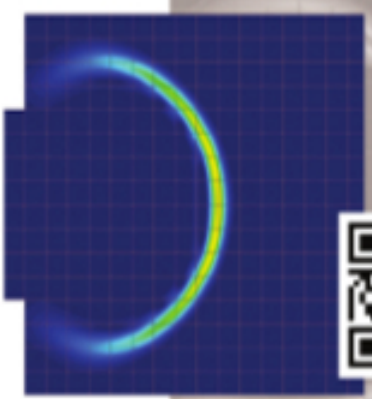
```
  // Initialize the weak formulation.
  double current_time = 0;

  CustomWeakFormHeatRK wf("Boundary_air", ALPHA, LAMBDA, HEATCAP, RHO,
                          &current_time, TEMP_INIT, T_FINAL);

  // Initialize boundary conditions.
  DefaultEssentialBCConst<double> bc_essential("Boundary_ground", TEMP_INIT);
  EssentialBCs<double> bcs(&bc_essential);

  // Create an H1 space with default shapeset.
  H1Space<double> space(&mesh, &bcs, P_INIT);
  int ndof = space.get_num_dofs();
  Hermes::Mixins::Loggable::Static::info("ndof = %d", ndof);

  // Initialize Runge-Kutta time stepping.
  RungeKutta<double> runge_kutta(&wf, &space, &bt);
```



FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Hermes Documentation

Release 3.1.0

hp-FEM group

August 03, 2014

1 Introduction **2**
1.1 Introduction . . . 2

2 Hermes Overview **3**
2.1 About Hermes . . . 3
2.2 Mathematical Background . . . 3
2.3 Citing Hermes . . . 8

3 Installation **11**
3.1 Linux . . . 11
3.2 Windows . . . 12
3.3 Mac OS . . . 18
3.4 Installation of Matrix Solvers . . . 19
3.5 Installation of ExodusII and NetCDF libraries . . . 25

4 Getting Started **28**
4.1 First steps . . . 28
4.2 Hermes typical example structure . . . 29
4.3 Introduction to advanced C++ object-oriented features . . . 35
4.4 Hermes C++ object model - deriving your own specialized classes . . . 35

5 Extended documentation **41**
5.1 Hermes Documentation overview . . . 41
5.2 Hermes Tutorial . . . 42
5.3 Examples Documentation . . . 42

6 Collaboration **43**
6.1 Collaboration via Github . . . 43

CHAPTER

ONE

INTRODUCTION

1.1 Introduction

Thank you for your interest in Hermes!

Hermes is a C++ library for rapid development of adaptive *hp*-FEM and *hp*-DG solvers, with emphasis on nonlinear, time-dependent, multi-physics problems.

This document is organized as follows:

- Section 1 provides general information about Hermes and the computational methods it uses, and how to install Hermes on various platforms.
- Section 2 is the Getting Started Guide - to get you started in no time.
- Section 3 is the extended documentation section, you will find developers documentation in Doxygen, as well as step-by-step user documentation in tutorials and advanced examples.
- Section 4 explains how to use Git and Github, and how you can contribute to the project if interested.

This document is under continuous development. If you find bugs, typos, dead links and such, please report them to the Hermes2D mailing list.

CHAPTER

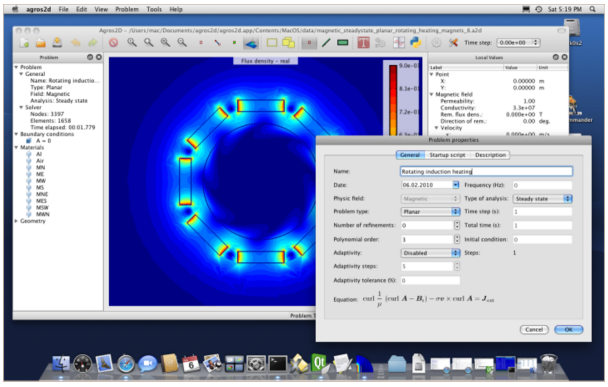
TWO

HERMES OVERVIEW

2.1 About Hermes

Hermes is a free C++ library for rapid development of adaptive *hp*-FEM and *hp*-DG solvers for partial differential equations (PDE) and multiphysics PDE systems. The development team now includes mostly the department of Theory of Electrical Engineering at the University of West Bohemia in Pilsen (contact: korous@rice.zcu.cz), in the past the main development was done by the hp-FEM group at the University of Nevada, Reno. Information about further collaborators from numerous places around the globe can be found at http://www.hpfem.org/citing/.

A standard way to use Hermes is to write short C++ user programs that use the functionality provided by the library, but for those who prefer to use a graphical interface, the group located at the University of West Bohemia also develops a graphical Engineering tool based on Hermes2D: Agros2D.



Hermes is loaded with modern finite element technology. We hope that you will enjoy the software and that you will find this documentation useful. In any case please let us know if you find mistakes or if you can suggest improvements to this documentation or to Hermes itself.

2.2 Mathematical Background

The main strength of Hermes is a modern adaptive higher-order finite element technology combined with very easy-to-use implementation approaches.

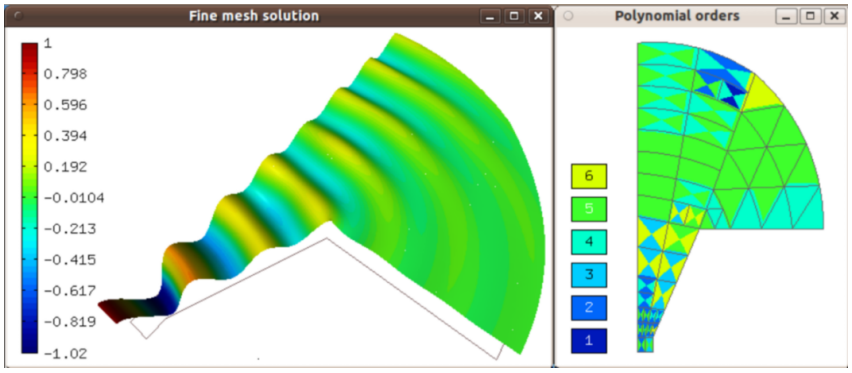
2.2.1 Features

- Curvilinear elements.
- Reduced mesh generation needs.

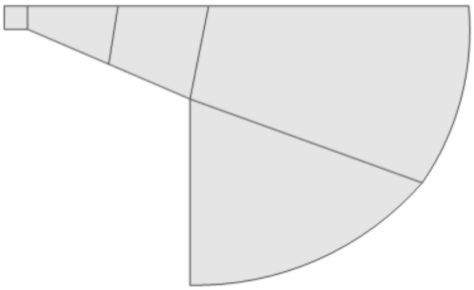
- Arbitrary-level hanging nodes.
- Scalar and vector-valued approximations.
- Advanced nonlinear solver capabilities.
- Exponential convergence of adaptive *hp*-FEM.
- Dozens of time-integration methods.
- Adaptivity with dynamical meshes for time-dependent problems.
- Adaptive multimesh *hp*-FEM for multiphysics coupled problems.
- Coupled problems solved together with hp-DG and hp-FEM.
- Calculations with physical quantities defined in different subdomains.

Some the above points are discussed in more detail below:

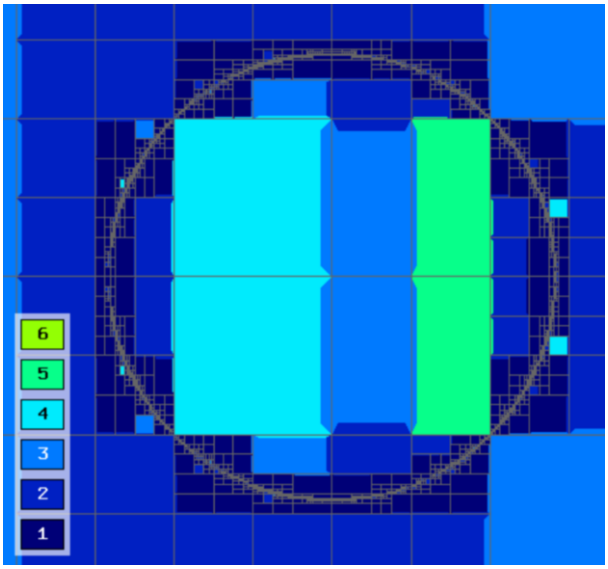
- **Curvilinear elements**: Approximating curved boundaries or material interfaces via small elements with straight edges belongs to history. It is much more efficient to employ curvilinear elements, such as in the following acoustics problem.



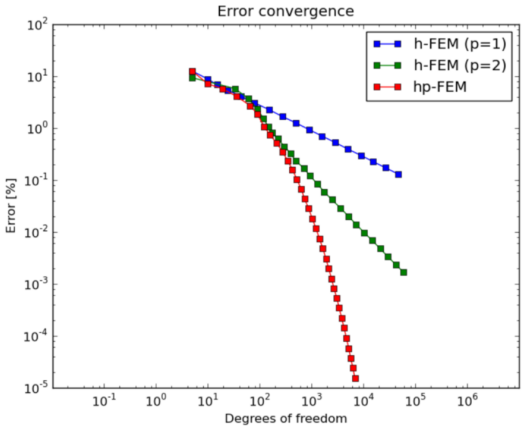
- **Reduced mesh generation needs**: The previous result was obtained with the mesh shown below, but we also provide support for traditional mesh generators including Triangle, CUBIT, GMSH.



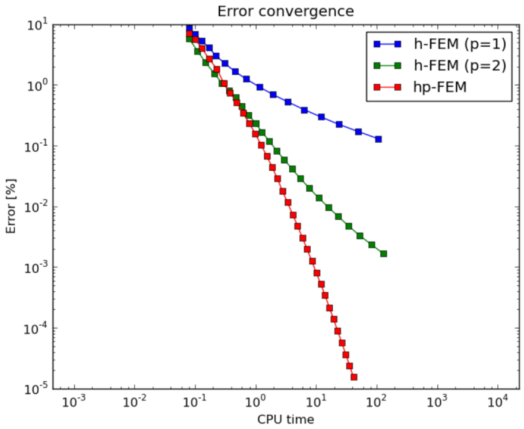
- **Arbitrary-level hanging nodes**: Hermes can handle irregular meshes with arbitrary-level hanging nodes. This makes adaptive algorithms much faster compared to algorithms that use regular meshes (without hanging nodes).



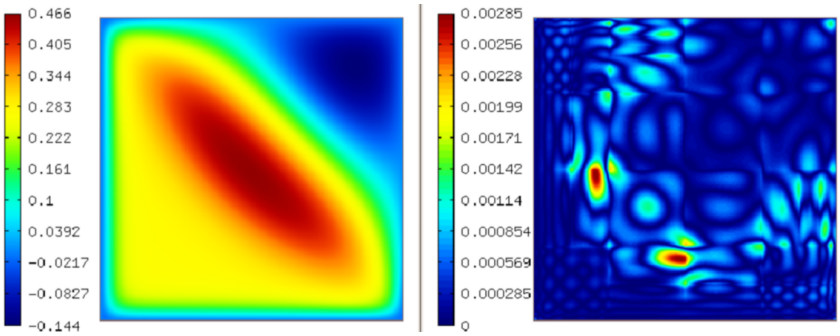
- **Exponential convergence of adaptive hp-FEM**: Many practitioners are skeptical about adaptive FEM because it makes computations slow. However, the exponential convergence of adaptive *hp*-FEM is very different from slow, algebraic convergence of standard low-order FEM. A typical comparison of adaptive low-order FEM and *hp*-FEM is shown below. Here *p=1* and *p=2* means linear and quadratic elements, respectively. The vertical axis shows the approximation error, the horizontal one the number of degrees of freedom (problem size).



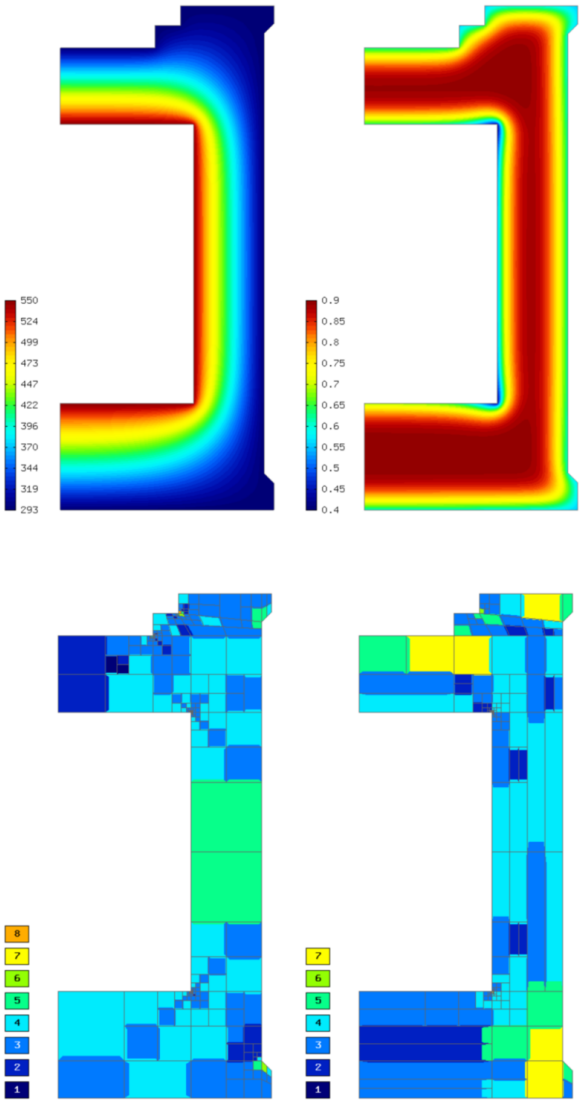
Same graphs as above but now in terms of CPU time:



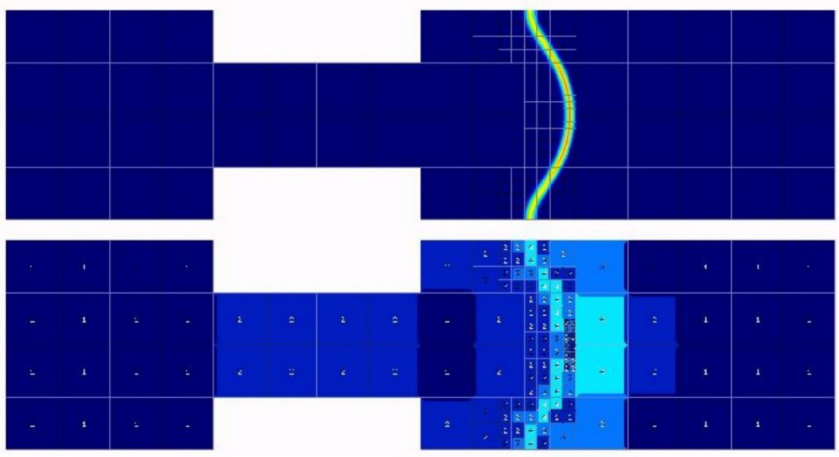
- **Dozens of time-integration methods**: Hermes has a unique way of using time-integration methods. More than 30 methods are readily available, including the most advanced adaptive implicit higher-order methods. The sample results below illustrate that it is highly recommended to take the time-discretization error seriously (below on the left). The reason is that it can easily be orders of magnitude larger than the error in space (below on the right).



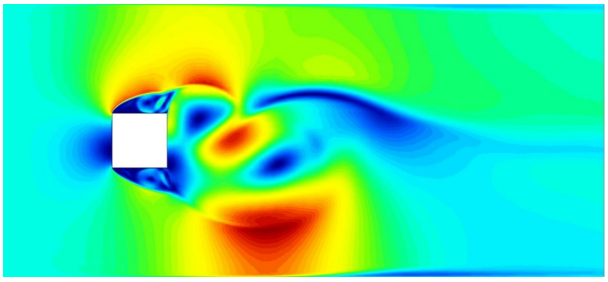
- **Multimesh hp-FEM**: Approximating different physical fields on the same mesh belongs to history. For a given solution component, just one finite element mesh can be optimal. Hermes uses an original adaptive multimesh *hp*-FEM technology to discretize any multiphysics problem *on multiple meshes in a monolithic fashion*. No error due to data transfer between various meshes is present. The following figure illustrates this on a coupled problem of heat and moisture transfer in massive concrete walls of a nuclear reactor vessel.



- **Dynamical meshes for time-dependent problems**: In time-dependent problems, different physical fields or solution components can be approximated on individual meshes that evolve in time independently of each other.



- **Wide applicability**: Hermes does not employ any error estimate or another technique that would limit its applicability to some particular class of PDE problems. It allows you to tackle an arbitrary PDE or multiphysics PDE system. Visit the hp-FEM group home page and the gallery to see examples.

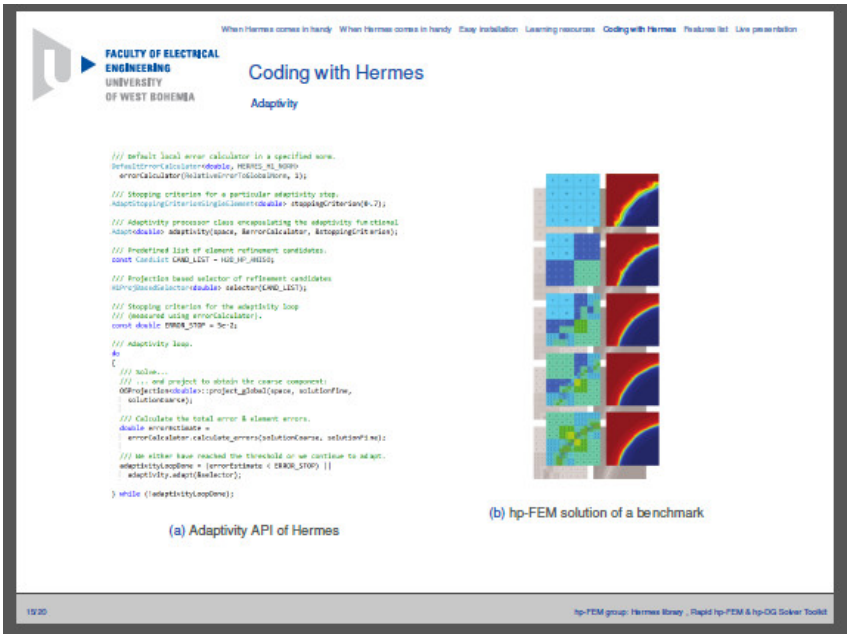


2.2.2 Implementation point of view

- OpenMP parallelization
- C++ templates for unified handling of real and complex problems
- Own OpenGL visualization & VTK outputs of meshes, spaces, solutions
- User-friendly written easy-to-grasp code
- XML, BSON save / load of the most important classes
- Solvers interfaces: UMFPACK, PARALUTION, PETSc, MUMPS, ...
- GMSH, ExodusII mesh formats
- Matlab, MatrixMarket matrix exports
- Well arranged doxygen documentation
- Exception safe API

2.2.3 Presentation about Hermes given at ESCO 2014 conference

A number of presentations about Hermes have been given.



Here is a link to one of them: https://github.com/hpfem/hermes/tree/master-3.1/doc/HermesPresentation.pdf.

2.3 Citing Hermes

If you use Hermes for your work, please be so kind to include some of the references below as appropriate.

Monographs:

```
@Book{Hermes-book1,
   author = {P. Solin, K. Segeth, I. Dolezel},
   title = {Higher-Order Finite Element Methods},
   publisher = {Chapman & Hall / CRC Press},
   year = {2003}
}

@Book{Hermes-book2,
   author = {P. Solin},
   title = {Partial Differential Equations and the Finite Element Method},
   publisher = {J. Wiley & Sons},
   year = {2005}
}
```

Reference to the Hermes open-source project:

```
@Manual{Hermes-project,
   title =  {Hermes - Higher-Order Modular Finite Element System (User's Guide)},
   author = {P. Solin et al.},
   url =    {http://hpfem.org/}
}
```

Underlying algorithms (hanging nodes, adaptivity, shape functions):

```
@ArticleHermes-time-integration,
  author = P. Solin, L. Korous,
  title = Adaptive Higher-Order Finite Element Methods for Transient PDE
  Problems Based on Embedded Higher-Order Implicit Runge-Kutta Methods,
  journal = J. Comput. Physics,
  year = 2011,
  status = accepted,
```

```
  volume = ,
  pages =


@ArticleHermes-hanging-nodes,
  author = P. Solin, J. Cerveny, I. Dolezel,
  title = Arbitrary-Level Hanging Nodes and Automatic Adaptivity
  in the hp-FEM,
  journal = Math. Comput. Simul.,
  year = 2008,
  volume = 77,
  pages = 117 - 132


@ArticleHermes-adaptivity,
  author = P. Solin, D. Andrs, J. Cerveny, M. Simko,
  title = PDE-Independent Adaptive hp-FEM Based on Hierarchic Extension of
  Finite Element Spaces,
  journal = J. Comput. Appl. Math.,
  year = 2010,
  volume = 233,
  pages = 3086-3094


@ArticleHermes-shape-functions,
  author = P. Solin, T. Vejchodsky,
  title = Higher-Order Finite Elements Based on Generalized Eigenfunctions of
  the Laplacian,
  journal = Int. J. Numer. Methods Engrg,
  year = 2007,
  volume = 73,
  pages = 1374 - 1394
```

Selected applications:

```
 @ArticleHermes-polymer-metal-composites,
   author = D. Pugal, P. Solin, K.J. Kim, A. Aabloo,
   title = Modeling Ionic Polymer-Metal Composites with Space-Time
   Adaptive Multimesh hp-FEM,
   journal = Communications in Computational Physics,
   year = 2011,
   status = accepted,
   volume = ,
   pages =


 @ArticleHermes-anisotropic-benchmarks,
   author = P. Solin, O. Certik, L. Korous,
   title = Three Anisotropic Benchmarks for Adaptive Finite Element Methods,
   journal = Appl. Math. Comput.,
   year = 2011,
   status = accepted,
   volume = ,
   pages =


 @ArticleHermes-richards,
   author = P. Solin, M. Kuraz,
   title = Solving the Nonstationary Richards Equation with Adaptive hp-FEM,
   journal = Advanced Water Resources,
   year = 2011,
   volume = 34,
```

```
   pages = 1062-1081


 @ArticleHermes-nuclear,
   author = L. Dubcova, P. Solin, G. Hansen, H. Park,
   title = Comparison of Multimesh hp-FEM to Interpolation and Projection Methods
   for Spatial Coupling of Reactor Thermal and Neutron Diffusion Calculations,
   journal = J. Comput. Physics,
   year = 2011,
   volume = 230,
   pages = 1182-1197


@ArticleHermes-heat-and-moisture,
  author = P. Solin, L. Dubcova, J. Kruis,
  title = Adaptive hp-FEM with Dynamical Meshes for Transient Heat and Moisture
  Transfer Problems,
  journal = J. Comput. Appl. Math,
  year = 2010,
  volume = 233,
  pages = 3103-3112


@ArticleHermes-thermoelasticity,
  author = P. Solin, J. Cerveny, L. Dubcova, D. Andrs,
  title = Monolithic Discretization of Linear Thermoelasticity Problems via
  Adaptive Multimesh hp-FEM,
  journal = J. Comput. Appl. Math,
  status = published online,
  doi = doi 10.1016/j.cam.2009.08.092,
  year = 2009


@ArticleHermes-electromagnetics,
  author = L. Dubcova, P. Solin, J. Cerveny, P. Kus,
  title = Space and Time Adaptive Two-Mesh hp-FEM for Transient Microwave Heating
  Problems,
  journal = Electromagnetics,
  year = 2010,
  volume = 30,
  pages = 23 - 40


@ArticleHermes-fluid-mechanics,
  author = P. Solin, J. Cerveny, L. Dubcova, I. Dolezel,
  title = Multi-Mesh hp-FEM for Thermally Conductive Incompressible Flow,
  journal = Proceedings of ECCOMAS Conference COUPLED PROBLEMS 2007 (M. Papadrakakis,
  E. Onate, B. Schrefler Eds.), CIMNE, Barcelona,
  year = 2007,
  pages = 677 - 680
```

Other papers that may be still closer to what you need can be found in the citing section of the hp-FEM group home page.

CHAPTER

THREE

INSTALLATION

3.1 Linux

3.1.1 Download and compilation

[NEW] - debian packages on Launchpad

You can download a package directly from https://launchpad.net/~lukas-korous/+archive/ubuntu/hermes.

The rest of the instructions here are for building Hermes from source.

If you are using a Debian-based system, install the (required) libraries first:

If you want to use fast saving / loading of Hermes entities, install

- BSON
  - Clone the BSON Mongo driver git repository from git@github.com:l-korous/mongo-c-driver.git (if you don't know how, here is a tip: Getting a Git Repository)
  - Compile and install using 'make install'

For thread caching memory allocator from Google, see

- TCMalloc
  - Get TCMalloc from the SVN repository at http://code.google.com/p/gperftools/source/checkout.
  - Make & install

To obtain the source code, clone the Git repository from Github:

```
git clone git://github.com/hpfem/hermes.git
```

These two repositories are synchronized. For more advanced users we recommend to create a free account at Github (if you do not have one yet), fork the Hermes repository, and then clone your Github copy of Hermes to your local computer. This will establish links between your local copy and the master repository, and you'll become part of the Hermes network at Github.

Once you have a local copy of the Hermes repository on your computer, change dir to hermes/. There you will find a CMake.vars.example.Linux file that looks like this:

```
# LINUX
  # On linux, there should be no need to set up *_ROOT directories, in the default settings, they
  # We mainly support gcc and CLang compilers with C++11 support.
```

```
  # BASIC CONFIGURATION

  # Global
# Generate static libs (instead of dynamic)
  set(HERMES_STATIC_LIBS NO)
  # Target path
  set(CMAKE_INSTALL_PREFIX "/usr/local")

  # Paths for compulsory dependencies
  set(XERCES_ROOT "/usr/local")
  set(XSD_ROOT "/usr/local")

  # HermesCommon

    # Release and debug versions
    set(HERMES_COMMON_DEBUG     YES)
    set(HERMES_COMMON_RELEASE   YES)
    ...
```

Copy this file to "CMake.vars" and set the variables according to your needs. After that, type:

```
cmake .
make
```

If you have more than one CPU, you can use "make -jN" where N is the number of CPUs of your computer.

3.1.2 Debugging with Eclipse

To use eclipse as debugger, in the root folder of the project:

```
mkdir eclipse_build
cd eclipse_build
cmake -G"Eclipse CDT4 - Unix Makefiles" -D CMAKE_BUILD_TYPE=Debug ../
```

In Eclipse:

- Import project using Menu File->Import
- Select General->Existing projects into workspace:
- Browse where your build tree is and select the root build tree directory.
- Keep "Copy projects into workspace" unchecked.

3.1.3 Install Hermes

```
make install
```

3.2 Windows

3.2.1 Download and compilation

[NEW] - prebuilt binaries

You can download both the dependency libraries and header files, as well as Hermes libraries and header files from https://github.com/l-korous/hermes-windows.

The rest of the instructions here are for building Hermes from source.

To obtain the source code, clone the Git repository from Github:

```
git clone git://github.com/hpfem/hermes.git
```

[IMPORTANT] As Hermes uses features of C++11 (such as initializer lists, nullptr_t, etc.), the only Visual Studio family compiler you can use is Visual Studio 2013 (both Express and 'Full' versions).

3.2.2 Dependency check-list - overview

- You need to install dependent libraries into either common directory, or separate directories. This directory, further called 'dependencies' (stands for the particular directory in the case of particular dependency), has to have three subdirectories as follows. The choice of having a single common directory, or separate ones is up to you.
  - 'dependencies'\include: Header files (*.h) of dependency libraries.
  - 'dependencies'\lib: Library files (*.lib) of dependency libraries.
  - 'dependencies'\bin: Binary modules (*.dll) of dependency libraries.
  - be sure to include a directory 'dependecies'\bin into the 'PATH' environment variable (you need to include all of them if you chose to have separate ones for various dependencies).
- For the 64-bit version, if you want to use it side-by-side to the 32-bit one, you can create a subdirectory 'x64' in the 'lib' folder where you will be putting the 64-bit dependency libraries.

3.2.3 Dependency check-list - 32-bit

This list works for 32-bit version of Hermes. See the section for 64-bit version if that is the one you are interested in. Please note that e.g. TCMalloc, BSON, UMFPACK are also 'optional', but to get the most performance out of Hermes, they are recommended.

- CMAKE
  - Download CMAKE installer(http://www.cmake.org/files/v2.8/cmake-2.8.3-win32-x86.exe) and install it.
- UMFPACK
  - MinGW used for compiling AMD and UMFPACK: Download MinGW.
  - after installing MinGW, add 'your-minGW-installation-directory'/bin to system PATH.
  - download latest SuiteSparse_config, AMD, and UMFPACK to A SINGLE PARENT DIRECTORY (this requirement is one of UMFPACK).
  - Add the following lines at the end of file SuiteSparse_config\SuiteSparse_config.mk:
    - CC = gcc
    - CXX = gcc
    - UMFPACK_CONFIG = -DNBLAS
    - RANLIB = echo
    - LIB = -lm
  - Open all files called 'Makefile' from all three directories and replace all ';' symbols in them with the Windows equivalent '&'

  - Copy SuiteSparse_config\SuiteSparse_config.h to 'include' directory
  - Copy SuiteSparse_config\libsuitesparseconfig.a to 'lib' directory and change its extension to Windows equivalent '.lib'.
  - Copy AMD\Include\amd.h, AMD\Include\amd_internal.h, and AMD\Lib\libamd.a to 'include', and 'lib' dependecy directories respectively. Change the libamd.a's extension to '.lib'
  - Copy UMFPACK\Include* to 'include'
  - Copy UMFPACK\Lib\libumfpack.a to 'lib' directory and change its extension to Windows equivalent '.lib'.
- XERCES
  - Download Xerces 3.1.1 source code from http://xerces.apache.org/xerces-c/download.cgi.
  - Build using your favorite compiler.
  - Copy all bin files to 'bin' dependencies directory
  - Copy all header files to 'include' dependencies directory
  - Copy the lib files to 'lib' dependencies directory
- XSD - Download XSD library from http://www.codesynthesis.com/download/xsd/3.3/windows/i686/xsd-3.3.0-i686-windows.zip, instructions how to build the library are available at http://wiki.codesynthesis.com/Using_XSD_with_Microsoft_Visual_Studio. - Copy all bin files to 'bin' dependencies directory - Copy all header files to 'include' dependencies directory
- OpenGL support (optional)
  - FREEGLUT
    - Download freeglut 2.4.0 (http://freeglut.sourceforge.net/) and unpack it.
    - Open the your_freeglut_2.4.0_root\freeglut.DSP file in Visual Studio and convert it to a newer format.
    - Compile Debug or Release version. Debug version is recommended in a case of debugging.
    - Copy 'freeglut.dll', 'freeglut.h', and 'freeglut.lib' to 'bin', 'include\GL', and 'lib' dependency directories, respectively/.
  - GLEW
    - Download glew Win32 precompiled binaries ver.1.5.4 (http://glew.sourceforge.net/) and unpack it.
    - Copy 'my_glew_root\bin\glew32.dll', 'my_glew_root\include\GL*.h', and 'my_glew_root\lib\glew32.lib' to 'bin', 'include\GL', and 'lib' dependency directories respectively.
  - PTHREAD(2.9.1)
  - Download appropriate files (ftp://sourceware.org/pub/pthreads-win32/prebuilt-dll-2-9-1-release/).
  - Copy 'dll\x86\pthreadVCE2.dll', 'include*.h' and 'lib\x86\pthreadVCE2.lib' to 'bin', 'include', and 'lib' dependecy directories respectively.
- The rest is optional. If a directive WITH_BSON is *not* used, this step including all sub-steps can be skipped and you can proceed to "**Building Hermes**".
  - MATIO (1.5.2)
    - Download HDF5 http://www.hdfgroup.org/ftp/HDF5/releases/hdf5-1.8.7/obtain5187.html.
    - Install HDF5, note the path (you will need it for MATIO)
    - Download MATIO from http://sourceforge.net/projects/matio/.
    - Open the sln file in the folder visual_studio

    * Add to the Include Directories under the libmatio project settings the directory where you installed HDF5's headers
    * Add to the Libraries Directories under the libmatio project settings the directory where you installed HDF5's libs
    * Add to the linker linking to "libszip.lib"
    * (Fix MATIO error) Open the file zconf.h and on the line 287 change #if 1 to #if 0.
    * build, copy visual_studio/*.h and src/*.h to 'include' folder, visual_studio/Release/libmatio.lib to 'lib', visual_studio/Release/libmatio.dll to 'bin' folders.

- BSON
    * Clone the BSON Mongo driver git repository from git@github.com:l-korous/mongo-c-driver.git (if you don't know how, here is a tip:Getting a Git Repository)
    * Download SCONS build tool from http://sourceforge.net/projects/scons/files/latest/download?source=files.
    * Install SCONS (you need to have PYTHON installed for that), run it (e.g. issuing C:Python27Scriptsscons.bat) in the BSON Mongo driver root directory
        · Use flags –m32 and –c99 ("C:Python27Scriptsscons.bat –c99 –m32")
    * Once compiled (should take seconds at most), copy src/bson.h to your 'include' dependency directory, bson.lib to 'lib', and bson.dll to 'bin' directories.
- TCMalloc
    * Get TCMalloc from the SVN repository at http://code.google.com/p/gperftools/source/checkout.
    * Open gperftools.sln in your Visual Studio, build the appropriate version (default works fine - just select Debug/Release)
    * Copy Win32"Release/Debug"libtcmalloc_minimal.dll to 'bin' dependency directory, Win32"Release/Debug"libtcmalloc_minimal.lib to 'lib' dependency directory
    * Copy the contents of src/google to 'include' dependency directory
- ExodusII
    * Download sources of version 4.9.3 (http://sourceforge.net/projects/exodusii/) and unpack 'exodusii'
    * Add the following line to the file 'my_exodusii_root\CMakeLists.txt' as:

      ```
      PROJECT(Exodusii)
      SET(NETCDF_INCLUDE_DIR "my_netcdf_root/libsrc4")
      # add this line;
      ```

      be sure to use a slash '/' instead of a backslash '\'.
    * Generate MSVC project files using CMAKE in command prompt as:

      ```
      cmake . -G "Visual Studio 9 2008"    # MSVC2008 user
      cmake . -G "Visual Studio 10"        # MSVC2010 user
      ```

      If you have Cygwin installed, make sure that you are using the windows version of cmake.
    * Open a SLN file 'my_exodusii_root/ExodusII.sln' in MSVC08/10
    * Switch to 'Release' version
    * Build a project 'exoIIv2c': this will create a LIB file in 'my_exodusii_root\cbind\Release'
    * Copy 'exoIIv2c.lib' to 'lib' dependency directory structure
    * Copy 'my_exodusii_root\cbind\include\exodusII.h and exodusII_ext.h' to 'include' dependency directory

  - CLAPACK
    * First, you need to install CLAPACK/CBLAS:
    * Download the file clapack-3.2.1-CMAKE.tgz from http://www.netlib.org/clapack/.
    * Use cmake to configure and build the debug version of clapack.
    * Copy '\clapack-3.2.1-CMAKE\BLAS\SRC\Debug\blas.lib', '\clapack-3.2.1-CMAKE\F2CLIBS\libf2c\Debug\libf2c.lib', and '\clapack-3.2.1-CMAKE\SRC\Debug\lapack.lib' to 'lib' dependency directory.
    * Copy the contains of '\clapack-3.2.1-CMAKE\INCLUDE\' to 'include' dependency directory.

3.2.4 Dependency check-list - 64-bit

Only the most important dependencies are described here for the 64-bit version. For all others, all you must do is compile the 64-bit version, or get it in any other way and link it to Hermes instead of the 32-bit one.

- CMAKE
  - Download CMAKE installer(http://www.cmake.org/files/v2.8/cmake-2.8.3-win32-x86.exe) and install it.
- PTHREAD(2.9.1)
  - Download appropriate files (ftp://sourceware.org/pub/pthreads-win32/prebuilt-dll-2-9-1-release/).
  - Copy 'dll\x64\pthreadVCE2.dll', 'include*.h' and 'lib\x64\pthreadVCE2.lib' to 'bin', 'include', and 'lib' dependecy directories respectively.
- UMFPACK
  - MinGW used for compiling AMD and UMFPACK: Download MinGW.
  - Just use 64-bit MinGW and provide the flag "-m64", otherwise it is the same as in Win32 version.
- XERCES
  - Download Xerces 3.1.1 source code from http://xerces.apache.org/xerces-c/download.cgi.
  - Build using your favorite compiler for 64-bit.
  - Copy all bin files to 'bin' dependencies directory
  - Copy all header files to 'include' dependencies directory
  - Copy the lib files to 'lib' dependencies directory
- XSD - Download XSD library from http://www.codesynthesis.com/download/xsd/3.3/windows/i686/xsd-3.3.0-i686-windows.zip, instructions how to build the library are available at http://wiki.codesynthesis.com/Using_XSD_with_Microsoft_Visual_Studio. - Build the x64 version - Copy all bin files to 'bin' dependencies directory - Copy all header files to 'include' dependencies directory
- OpenGL support (optional)
  - FREEGLUT
    * Download freeglut 2.4.0 (http://freeglut.sourceforge.net/) and unpack it.
    * Open the your_freeglut_2.4.0_root\freeglut.DSP file in Visual Studio and convert it to a newer format.
    * Compile Debug or Release version (x64 platform). Debug version is recommended in a case of debugging.

        * Copy 'freeglut.dll', 'freeglut.h', and 'freeglut.lib' to 'bin', 'include\GL', and 'lib' dependency directories, respectively/.
    - GLEW
        * Download glew x64 precompiled binaries (http://glew.sourceforge.net/) and unpack it.
        * Copy 'my_glew_root\bin\glew32.dll', 'my_glew_root\include\GL*.h', and 'my_glew_root\lib\glew32.lib' to 'bin', 'include\GL', and 'lib' dependency directories respectively.
- The rest is optional. If a directive WITH_BSON is *not* used, this step including all sub-steps can be skipped and you can proceed to "**Building Hermes**".
    - MATIO (1.5.2)
        * Just follow the 32-bit version instructions and download HDF5 for x64, and also when building MATIO, build the x64 version.
    - TCMalloc
        * Get TCMalloc from the SVN repository at http://code.google.com/p/gperftools/source/checkout.
        * Open gperftools.sln in your Visual Studio, build the appropriate version (default works fine - just select Debug/Release)
        * Copy x64"Release/Debug"libtcmalloc_minimal.dll to 'bin' dependency directory, x64"Release/Debug"libtcmalloc_minimal.lib to 'lib' dependency directory
        * Copy the contents of src/google to 'include' dependency directory
    - BSON
        * Clone the BSON Mongo driver git repository from git@github.com:l-korous/mongo-c-driver.git (if you don't know how, here is a tip:Getting a Git Repository)
        * Download SCONS build tool from http://sourceforge.net/projects/scons/files/latest/download?source=files.
        * Install SCONS (you need to have PYTHON installed for that), run it (e.g. issuing C:Python27Scriptsscons.bat) in the BSON Mongo driver root directory
            · Use the flag –c99 ("C:Python27Scriptsscons.bat –c99")
        * Once compiled (should take seconds at most), copy src/bson.h to your 'include' dependency directory, bson.lib to 'lib', and bson.dll to 'bin' directories.

3.2.5 Building Hermes

In order to build the library and examples, you need to:

- Prepare dependecy libraries, see 'Dependency Check-list'.
- Copy the file 'CMake.vars.example.Windows' to 'CMake.vars'. The file contains settings for the project.
- In the root Hermes directory, generate project files by running CMAKE from a command prompt:

  ```
  cmake . -G "Visual Studio 12"      # MSVC2013 as the generator
  ```

  If you have Cygwin installed, your might have an error "Could not create named generator Visual Studio 12". This is because your cmake path is contaminated by Cygwin's cmake. Try to use absolute path for windows cmake.exe.
- Open the SLN file 'hermes.sln' and build Hermes.

3.2.6 Using Hermes

In order to use Hermes in your project, you need to do the following steps. Steps has 5, 6, and 7 to be repeated for every configuration, i.e., Debug, Release. Except the step 7b, this can be done easily by setting the drop-down Configuration to 'All configurations' in the Project Property dialog.

- Prepare Hermes to be buildable by MSVC, see 'Building Hermes'.
- Create your project in MSVC. Set the project to be an empty Win32 console project.
- Add directories 'dependencies\lib' to additional library directories (<right click on your project>\Properties\Configuration Properties\Linker\Additional Library Directories).
- Add also the directory where you copied Hermes libraries to as an additional library directory. This would probably be the variable CMAKE_INSTALL_PREFIX in your CMake.vars file.
- Add 'include "hermes2d.h"', make sure that your CMAKE_INSTALL_PREFIX is among Include Directories settings in your compiler.
- Add the dependencies\include directory (and possibly other directories where you copied dependency headers) using
  - Project -> Properties -> Configuration Properties -> VC++ Directories -> Include Directories
- Deny (Ignore) warnings that are not indicating anything dangerous:
  - Ignore warnings about STL in DLL by denying a warning 4251 (<right click on your project>\Properties\Configuration Properties\C/C++\Advanced\Disable Specific Warnings, enter 4251).
  - Ignore warnings about standard functions that are not safe (<right click on your project>\Properties\Configuration Properties\C/C++\Preprocessor\Preprocessor Definitions, add _CRT_SECURE_NO_WARNINGS).
  - Also ignore any template instantiation warnings
- Resolve unresolved linker error in Xerces - http://stackoverflow.com/questions/10506582/xerces-c-unresolved-linker-error

3.3 Mac OS

3.3.1 Download and compilation

Known issues: Hermes has built-in OpenGL visualization based on FreeGlut, but this package comes with certain installation difficulties. If you encounter Glut-related problems, set H2D_WITH_GLUT to NO in Cmake.vars, build Hermes without Glut, and use VTK output for visualization.

Step 1: Make sure you have XCode installed. This should be on the installation disks which came with your Mac. XCode contains the GNU compilers, make and many other things which are required to build Hermes.

Step 3: Install the following libraries and applications: Suitesparse, glew, cmake, git. If you don't already have these on your Mac, then the easiest way to get them is to use MacPorts (which is an application which allows you to easily install and manage UNIX libraries and applications on your Mac) by doing the following:

1. Download and install MacPorts from http://www.macports.org/install.php.
2. Do 'sudo port install suitesparse glew'.
3. If you don't already have git installed, do 'sudo port install git'.
4. If you don't already have cmake installed, do 'sudo port install cmake'.

Step 4: Get the Hermes source code as described at the beginning of the Linux section above. Change to the directory where you want to download the Hermes source and clone the git repository either from the hpfem.org server:

```
git clone `<http://git.hpfem.org/git/hermes.git>`_.
```

or from Github:

```
git clone git://github.com/hpfem/hermes.git
```

These two repositories are synchronized. For more advanced users we recommend to create a free account at Github (if you do not have one yet), fork the Hermes repository, and then clone your Github copy of Hermes to your local computer. This will establish links between your local copy and the master repository, and you'll become part of the Hermes network at Github.

Step 5: Configure and build Hermes by changing dir to 'hermes/', and then typing 'cmake .' and 'make'. If you have more than one CPU, you can use 'make -jN' where N is the number of CPUs of your computer. To set the location where Hermes will be installed, pass the -DCMAKE_INSTALL_PREFIX=<your location> flag to cmake (i.e. to install in /usr/local, replace the cmake command above with 'cmake -DCMAKE_INSTALL_PREFIX=/usr/local .').

Step 6: Install Hermes by doing 'make install'.

3.3.2 More options

You can turn on and off various components to build, just create the CMake.vars file and add the following:

```
set(H2D_WITH_GLUT NO)
```

(and any other option that you would like to change, see CMakeLists.txt for the whole list).

For development, it is good to say (in global CMake.vars):

```
set(DEBUG YES) to compile debug versions
set(RELEASE YES) to compile release versions
```

Then type:

```
make debug    (to build debug versions)
make release  (to build release versions)
```

3.4 Installation of Matrix Solvers

3.4.1 Mumps

Linux

Using standard Debian packages

For sequential version, install packages *libmumps-seq-4.9.2* and *libmumps-seq-dev*. For parallel version, install *libmumps-4.9.2* and *libmumps-dev*. Newer versions may be available. In Ubuntu 6.06 (Dapper) or newer, you can use the Synaptic package manager for that, or type:

```
sudo apt-get install libmumps-seq-4.9.2 libmumps-seq-dev
```

for the sequential version and

```
sudo apt-get install libmumps-4.9.2 libmumps-dev
```

for the parallel one.

Now go to the directory with Hermes. Create the file CMake.vars with the following line (or append to the existing one):

```
set(WITH_MUMPS YES)
```

Finally execute:

```
rm CMakeCache.txt
cmake .
make
```

Find more about *Using MUMPS in Hermes*.

Windows MSVC

Installation of MUMPS using MSVC is rather easy:

- preparation
  - download MUMPS from http://mumps.enseeiht.fr/MUMPS_4.10.0.tar.gz (if the link does not work, look for 4.10 version of MUMPS)
  - download WinMUMPS utility from http://sourceforge.net/projects/winmumps/.
  - download a Fortran compiler (e.g. http://software.intel.com/en-us/intel-fortran-studio-xe-evaluation-options).
  - download BLAS (Debug/Release, static/dynamic, 32-bit/64-bit as you like) from http://icl.cs.utk.edu/lapack-for-windows/lapack/index.html#libraries.
  - you have to have Visual Studio version >= 2008
  - you have to have Python 2.6 or 2.7 available
- installation
  - copy the downloaded BLAS library to 'bin' and 'lib' dependencies directory respectively. Please not that the name of the static library (static part of the dynamic library) should be either blas.lib, or libblas.lib
  - unzip MUMPS and WinMUMPS
  - execute "python winmumps_generator.py" from WinMUMPS providing the options needed (type -h for help)
  - this will generate several C++ project files (.vcxproj) and several Fortran project files (.vfproj)
  - open one of them and add the rest to the solution
  - build the solution in your desired configuration (Win32 or x64, Debug or Release)
  - this will place the libraries into MUMPS_4.10.0libDebugWin32(or alternatively for other configurations)
  - copy those to 'lib' dependencies directory
  - copy the contents of MUMPS_4.10.0includeto 'include' dependencies directory

Mac OS

Help needed!

Using MUMPS in Hermes

After the installation has been completed, you may select `SOLVER_MUMPS` as the matrix solver for your finite element problem, as detailed in the *Poisson tutorial* (in tutorial found on Hermes website), or use it just to solve a standalone matrix problem $Ax = b$.

3.4.2 PETSc

Linux

Using standard Debian packages

Install packages *libpetsc3.1* and *libpetsc3.1-dev*. Newer version may be available. In Ubuntu 6.06 (Dapper) or newer, you can use the Synaptic package manager for that, or type:

```
sudo apt-get install libpetsc3.1 libpetsc3.1-dev
```

Now go to the directory with Hermes. Create the file CMake.vars with the following line (or append to the existing one):

```
set(WITH_PETSC YES)
```

Finally execute:

```
rm CMakeCache.txt
cmake .
make
```

Find more about *Using PETSC in Hermes*.

Windows MSVC

http://www.mcs.anl.gov/petsc/petsc-as/documentation/installation.html#Windows

Mac OS

http://petsc.darwinports.com/

Using PETSC in Hermes

You may now select `SOLVER_PETSC` as the matrix solver for your finite element problem, as detailed in the Poisson tutorial, or use it just to solve a standalone matrix problem $Ax = b$ as in the Using Matrix Solvers tutorial.

3.4.3 SuperLU

Hermes currently supports two versions of the SuperLU library - the sequential one and the multithreaded one. Support for the MPI version will be added in the future. Please visit http://crd.lbl.gov/~xiaoye/SuperLU/ for more information about the library.

Linux

Using standard Debian packages

Install the *libsuperlu3* and *libsuperlu3-dev* packages. In Ubuntu 6.06 (Dapper) or newer, you can use the Synaptic package manager for that, or type:

```
sudo apt-get install libsuperlu3 libsuperlu3-dev
```

Now go to the directory with Hermes. Create the file CMake.vars with the following lines (or append to the existing one):

```
set(WITH_SUPERLU YES)
set(SUPERLU_ROOT ~/solvers/superlu_mt) #(or your own installation destination)
set(SUPERLU_MT   NO)
```

Finally execute:

```
rm CMakeCache.txt
cmake .
make
```

Find more about *Using SUPERLU in Hermes*.

Windows MSVC

http://crd.lbl.gov/~xiaoye/SuperLU/faq.html

MAC OS

http://www.bleedingmind.com/index.php/2010/07/31/compiling-superlu-on-os-x/

Using SUPERLU in Hermes

You may now select `SOLVER_SUPERLU` as the matrix solver for your finite element problem, as detailed in the Poisson tutorial, or use it just to solve a standalone matrix problem $Ax = b$ as in the Using Matrix Solvers tutorial.

3.4.4 Trilinos

Linux

Using standard Debian packages

Install packages *libtrilinos* and *libtrilinos-dev*. In Ubuntu 6.06 (Dapper) or newer, you can use the Synaptic package manager for that, or type:

```
sudo apt-get install libtrilinos libtrilinos-dev
```

Now go to the directory with Hermes. Create the file CMake.vars with the following line (or append to the existing one):

```
set(WITH_TRILINOS YES)
```

Finally execute:

```
rm CMakeCache.txt
cmake .
make
```

Find more about *Using TRILINOS in Hermes*.

Windows

First of all - to build Trilinos, one needs LAPACK (CLAPACK) (see the optional package in the library installation documentation).

Download the sources for the latest version from the Trilinos page and unpack them in some temporary directory.

Go to the Trilinos source directory.

In the following, replace {TPL_LAPACK_LIBRARIES}, {TPL_BLAS_LIBRARIES} with the full path to your lapack.lib and blas.lib without any quotes.
Also, replace {CMAKE_INSTALL_PREFIX} with either your dependency root, or any other folder where you want to install Trilinos packages.

```
mkdir build_dir
cd build_dir
cmake \
 -D CMAKE_BUILD_TYPE:STRING=DEBUG \
 -D TPL_LAPACK_LIBRARIES:FILEPATH=d:\\hpfem\\hermes\\dependencies\\lib\\lapack.lib \
 -D TPL_BLAS_LIBRARIES:FILEPATH=d:\\hpfem\\hermes\\dependencies\\lib\\blas.lib \
 -D CMAKE_Fortran_FLAGS:STRING="-fPIC" \
 -D Trilinos_ENABLE_ALL_PACKAGES:BOOL=OFF \
 -D Trilinos_ENABLE_Teuchos:BOOL=ON \
 -D Teuchos_ENABLE_TESTS:STRING=OFF \
 -D Teuchos_ENABLE_EXAMPLES:STRING=OFF \
 -D Trilinos_ENABLE_Epetra:BOOL=ON \
 -D Epetra_ENABLE_TESTS:STRING=OFF \
 -D Epetra_ENABLE_EXAMPLES:STRING=OFF \
 -D Trilinos_ENABLE_EpetraExt:BOOL=ON \
 -D EpetraExt_ENABLE_TESTS:STRING=OFF \
 -D EpetraExt_ENABLE_EXAMPLES:STRING=OFF \
 -D Trilinos_ENABLE_AztecOO:BOOL=ON \
 -D AztecOO_ENABLE_TESTS:STRING=OFF \
 -D AztecOO_ENABLE_EXAMPLES:STRING=OFF \
 -D Trilinos_ENABLE_Ifpack:BOOL=ON \
 -D Ifpack_ENABLE_TESTS:STRING=OFF \
 -D Ifpack_ENABLE_EXAMPLES:STRING=OFF \
 -D Trilinos_ENABLE_ML:BOOL=ON \
 -D ML_ENABLE_TESTS:STRING=OFF \
 -D ML_ENABLE_EXAMPLES:STRING=OFF \
 -D Trilinos_ENABLE_OpenMP:BOOL=ON \
 -D Trilinos_ENABLE_Amesos:BOOL=ON \
 -D Amesos_ENABLE_TESTS:STRING=OFF \
 -D Amesos_ENABLE_EXAMPLES:STRING=OFF \
 -D Trilinos_ENABLE_NOX:BOOL=ON \
 -D NOX_ENABLE_TESTS:STRING=OFF \
 -D NOX_ENABLE_EXAMPLES:STRING=OFF \
 -D Trilinos_ENABLE_Anasazi:BOOL=ON \
 -D Anasazi_ENABLE_TESTS:STRING=OFF \
 -D Anasazi_ENABLE_EXAMPLES:STRING=OFF \
 -D Trilinos_ENABLE_Komplex:BOOL=ON \
 -D Komplex_ENABLE_TESTS:STRING=OFF \
 -D Komplex_ENABLE_EXAMPLES:STRING=OFF \
 -D Trilinos_ENABLE_TESTS:BOOL=OFF \
 -D DART_TESTING_TIMEOUT:STRING=600 \
 -D CMAKE_INSTALL_PREFIX:STRING=/d/hpfem/hermes/dependencies \
 ..
```

Build the Trilinos solution.
Go up to the Trilinos source directory and edit the cmake_install.cmake file. Change::

SET(CMAKE_INSTALL_CONFIG_NAME “Release”)

for:

```
SET(CMAKE_INSTALL_CONFIG_NAME "Debug")
```

Install Trilinos into the path specified by the {CMAKE_INSTALL_PREFIX} variable by running:

```
cmake -P cmake_install.cmake
```

You may also need to create a dummy file "unistd.h" in the include folder under dependencies folder. This header is not present in certain versions of Microsoft C library. Go to the directory with Hermes. Add the following lines into CMake.vars:

```
set(WITH_TRILINOS YES)
set(TRILINOS_ROOT {CMAKE_INSTALL_PREFIX})
```

again, replace {CMAKE_INSTALL_PREFIX} with the folder where you installed Trilinos.

Find more about *Using TRILINOS in Hermes*.

MAC OS

In preparation.

Using TRILINOS in Hermes

You may now select either `SOLVER_AMESOS` as the direct matrix solver or `SOLVER_AZTECOO` as the iterative matrix solver for your finite element problem, as detailed in the Poisson tutorial, or use it just to solve a standalone matrix problem $Ax = b$ as in the Using Matrix Solvers tutorial. Note that Trilinos is also required for using the advanced nonlinear solver `NOX` (see e.g. the Trilinos - Nonlinear tutorial).

3.4.5 UMFpack

Linux

Using standard Debian packages

Install the *libsuitesparse-metis-3.1.0* and *libsuitesparse-dev* packages. In Ubuntu 9.10 (Karmic) or newer you can use the Synaptic package manager for that, or type:

```
sudo apt-get install libsuitesparse-metis-3.1.0 libsuitesparse-dev
```

Now go to the directory with Hermes. Create the file CMake.vars with the following lines (or append to the existing one):

```
set(WITH_UMFPACK YES)
```

and execute:

```
rm CMakeCache.txt
cmake .
make
```

Find more about *Using UMFPACK in Hermes*.

Windows (Cygwin, MinGW, MSVC)

http://matrixprogramming.com/2008/03/umfpack

Mac OS

http://mywiki-science.wikispaces.com/UMFPACK

Using UMFPACK in Hermes

After the installation has been completed, you may select `SOLVER_UMFPACK` as the matrix solver for your finite element problems:

```
HermesCommonApi.set_integral_param_value(matrixSolverType, SOLVER_UMFPACK);
```

3.4.6 PARALUTION

1. Download PARALUTION from http://www.paralution.com/.
2. Compile, build PARALUTION, and copy headers, and libraries so that Hermes's CMake system can find them (as with other dependencies)
   (a) On Linux, installing PARALUTION to default install directories is sufficient, on Windows some paths have to be set
3. In your CMake.vars file (or directly in CMakeLists.txt) in the root of Hermes (see step 0) add "set(WITH_PARALUTION YES)"
   (a) It is on by default, so by default one has to include PARALUTION to build Hermes
4. That is it, build Hermes, it will automatically link to PARALUTION, include headers and make it usable.

3.4.7 How to use PARALUTION

5. Read the doxygen manual of the classes
   (a) Hermes::Algebra::ParalutionMatrix
   (b) Hermes::Algebra::ParalutionVector
   (c) Hermes::Preconditioners::ParalutionPrecond
   (d) Hermes::Solvers::IterativeParalutionLinearMatrixSolver
   (e) Hermes::Solvers::AMGParalutionLinearMatrixSolver
   (f) and all classes that these inherit from / use
6. If you want to see Hermes & PARALUTION readily work together, take any test example in the /hermes2d folder in the Hermes root and add one of these lines at the beginning of your main()
   (a) HermesCommonApi.set_integral_param_value(matrixSolverType, SOLVER_PARALUTION_ITERATIVE); // to use iterative solver
   (b) HermesCommonApi.set_integral_param_value(matrixSolverType, SOLVER_PARALUTION_AMG); // to use AMG solver
7. Solver classes of Hermes (NewtonSolver, PicardSolver, LinearSolver, ...) will then take this API setting into account and use PARALUTION as the matrix solver.

3.5 Installation of ExodusII and NetCDF libraries

ExodusII and NetCDF libraries are necessary to read Cubit files in Hermes.

3.5.1 Installing ExodusII

To install the ExodusII library, insert the following commands in your command-line terminal (be root user for system wide installation):

Step 1)

```
cd /
```

Step 2)

64bit:

```
wget http://hpfem.org/downloads/exodusii-bin-amd64-4.81.tar.gz
```

32bit:

```
wget http://hpfem.org/downloads/exodusii-bin-i686-4.81.tar.gz
```

Step 3)

```
tar xvzf exodusii-bin-*.tar.gz
```

Step 4)

```
rm exodusii-bin-*.tar.gz
```

The latter steps will unpack the binary package of ExodusII into /opt/packages/exodusii. If you do not like the location, you may change it to whatever you like, just remember to also adjust the "CMake" variables accordingly (see below).

To use ExodusII support in Hermes2d or 3d: - clone the Hermes repo (if you haven't done so yet), and in your "CMake.vars" file (found in your Hermes directory) add the following lines (AFTER you are done installing BOTH [ExodusII and NetCDF] packages):

```
SET(WITH_EXODUSII YES)
SET(EXODUSII_ROOT /opt/packages/exodusii)
SET(NETCDF_ROOT /opt/packages/netcdf)
```

After you are done adding these lines to your "CMake.vars" file, go back to your command-line terminal and type:

```
cmake .
make
```

Congratulations (assuming everything worked out) you just installed the ExodusII library!

3.5.2 Installing NetCDF

To install the NetCDF library, insert the following commands in your command-line terminal (be root user for system wide installation):

Step 1)

```
cd /
```

Step 2)

64bit:

```
wget http://hpfem.org/downloads/netcdf-bin-amd64-4.0.1.tar.gz
```

32bit:

```
wget http://hpfem.org/downloads/netcdf-bin-i686-4.0.1.tar.gz
```

Step 3)

```
tar xvzf netcdf-bin-*.tar.gz
```

Step 4)

```
rm netcdf-bin-*.tar.gz
```

The latter steps will unpack the binary package of NetCDF into /opt/packages/netcdf. If you do not like the location, you may change it to whatever you like, just remember to also adjust the "CMake" variables accordingly (see below).

To use ExodusII support in Hermes2d or 3d: - clone the Hermes repo (if you haven't done so yet), and in your "CMake.vars" file (found in your Hermes directory) add the following lines (AFTER you are done installing BOTH [ExodusII and NetCDF] packages):

```
SET(WITH_EXODUSII YES)
SET(EXODUSII_ROOT /opt/packages/exodusii)
SET(NETCDF_ROOT /opt/packages/netcdf)
```

After you are done adding these lines to your "CMake.vars" file, go back to your command-line terminal and type:

```
cmake .
make
```

Congratulations (assuming everything worked out) you just installed the NetCDF library!

Note: The binary packages were prepared on Ubuntu 9.10 (64bit and 32bit); they may or may not work on other 64bit and 32bit machines.

CHAPTER

FOUR

GETTING STARTED

4.1 First steps

After a successful compilation of Hermes library, you can install it by typing 'make install' (probably with sudo prefix). This will install Hermes to the target directory of your previous choice (or the default ones).

You can see what is being installed right on the screen (either in bash in Linux, or in Visual Studio output window on Windows). It is:

```
the two dynamically-linked libraries ('hermes_common' and 'hermes2d'), these will go to the 'lib'
two include directories ('hermes_common' and 'hermes2d'), these will go to the 'include' subdirect
```

The fact that they are linked dynamically is more important on Windows, where you have to **set your PATH environment variable** to point to the directory where you installed it. If you do not know how to do that, google it.

The recommended way of learning how to use Hermes for your purposes is to take a look at the small collection of what we call 'test examples' right there in the directory.

If you chose not to build them in cmake (using **set(H2D_WITH_TEST_EXAMPLES NO)**), change this, and rebuild the library to see them.

If something is not clear enough from the comments in the code, please see the section **Hermes typical example structure**.

These examples (located in hermes2d/test_examples) are:

```
00-quickShow
01-poisson
02-poisson-newton
03-navier-stokes
04-complex-adapt
05-hcurl-adapt
06-system-adapt
07-newton-heat-rk
08-nonlinearity
09-trilinos-nonlinear
10-linear-advection-dg-adapt
11-transient-adapt
12-picard
13-FCT
14-error-calculation
15-adaptivity-matrix-reuse-simple
16-adaptivity-matrix-reuse-layer-interior
```

And these examples are well-documented showcase examples of how to use Hermes.

Take a look at the next section documenting a typical example structure **Hermes typical example structure**.

4.2 Hermes typical example structure

- a nice thing about Hermes is that it follows the math tightly, so the steps taken in solving the example with Hermes correspond to those taken in theory.

A beginning of each example can look like this:

```
// Include the main Hermes2D header.
include "hermes2d.h"

// Two basic namespaces.
using namespace Hermes;
using namespace Hermes::Hermes2D;

// For adaptivity.
using namespace Hermes::Hermes2D::RefinementSelectors;

// For visualization.
using namespace Hermes::Hermes2D::Views;
```

4.2.1 Mesh

First part one needs to handle is the computational mesh, typically the following would be used:

```
// Shared pointers are used for easier memory handling.
MeshSharedPtr mesh(new Mesh);

// Either: Native Hermes mesh format.
MeshReaderH2D mloader;
mloader.load("domain.mesh", mesh);

// Or: XML Hermes mesh format.
MeshReaderH2DXML mloader;
mloader.load("domain.xml", mesh);

// Or: BSON (Binary JSON) format for fast binary load/save - used primarily in Agros.
MeshReaderH2DBSON mloader;
mloader.load("domain.bson", mesh);
```

More about meshes can be found in the 'hermes-tutorial' documentation, section 'A-linear', chapter '01-mesh' and in the **Doxygen documentation**.

4.2.2 Space

Secondly, the Finite Element space must be set on the computational mesh. One of the following is typically used (including setting of Dirichlet boundary conditions):

```
// H1 Space.
// Polynomial order.
int POLYNOMIAL_ORDER = 3;

// Initialize boundary conditions.
// This is a custom (derived) boundary condition. More about this in the section
// 'Object model - deriving your own specialized classes'.
CustomDirichletCondition bc_essential(
  std::vector<std::string>({"Bottom", "Inner", "Outer", "Left"}),
  BDY_A_PARAM, BDY_B_PARAM, BDY_C_PARAM);

// Initialize the container to pass the boundary conditions to the Space.
EssentialBCs<double> bcs(&bc_essential);
```

```
// Create an H1 space.
SpaceSharedPtr<double> space(new H1Space<double>(mesh, &bcs, POLYNOMIAL_ORDER));

// HCurl Space.
// Polynomial order.
int POLYNOMIAL_ORDER = 5;

// Initialize boundary conditions.
Hermes::Hermes2D::DefaultEssentialBCConst<std::complex<double> > bc_essential
(std::vector<std::string>({"Corner_horizontal", "Corner_vertical"}), 0);
EssentialBCs<std::complex<double> > bcs(&bc_essential);

// Create an Hcurl space.
SpaceSharedPtr<std::complex<double> > space(new HcurlSpace<std::complex<double> >(mesh, &bcs, POL

// HDiv Space. This example does not use any Dirichlet boundary conditions.
int POLYNOMIAL_ORDER = 2;
SpaceSharedPtr<double> space(new HdivSpace<double>(mesh, POLYNOMIAL_ORDER));

// L2 Space. This Space does not take any boundary conditions which corresponds to the
// fact that the FE space is a space of discontinuous functions.
// If we for example use polynomial order = 0, we use just piecewise
// constant basis functions.
SpaceSharedPtr<double> space(new L2Space<double>(mesh, 0));
```

More about spaces can be found in the 'hermes-tutorial' documentation, section 'A-linear', chapter '02-space' and in the **Doxygen documentation**.

More about Dirichlet boundary conditions can be found in the 'hermes-tutorial' documentation, section 'A-linear', chapter '04-bc-dirichlet', and for defining a non-constant custom boundary condition, see the chapter '07-general'.

4.2.3 Weak formulation

When we already have a mesh and a space, we have to know what equations we will be solving on those. And that is where the weak formulation comes to the light. Of course, there is a vast mathematical background of differential equations, their weak solutions, Sobolev spaces, etc., but we assume of those, our users already have a good knowledge. Right here we are concerned with the implementation. A typical creation of a weak formulation for the use with Hermes might look like this:

```
// Initialize the weak formulation (strictly speaking, shared pointer to the weak formulation)
// This is a weak formulation for linear elasticity, with custom
// parameters of the constructor.
// There is a lot of documentation for using some predefined
// weak forms, as well as creating your own. See the info below.
WeakFormSharedPtr<double> wf(new CustomWeakFormLinearElasticity(E, nu, rho*g1, "Top", f0, f1));
```

More about a typical basic weak form can be found in the 'hermes-tutorial' documentation, section 'A-linear', chapter '03-poisson'.

More about creating a custom weak form can be found in other tutorial examples. One always needs to subclass the Hermes::Hermes2D::WeakForm<Scalar> class template. For defining custom forms (integrals), one needs to subclass templates Hermes::Hermes2D::MatrixFormVol<Scalar>, MatrixFormSurf<Scalar>, VectorFormVol<Scalar>, VectorFormSurf<Scalar>. A typical constructor of a derived class:

```
template<> DefaultMatrixFormVol<std::complex<double> >::DefaultMatrixFormVol
  (int i, int j, std::string area, Hermes2DFunction<std::complex<double> >* coeff,
  SymFlag sym, GeomType gt)
  : MatrixFormVol<std::complex<double> >(i, j, area, sym), coeff(coeff), gt(gt)
{
  // If coeff is HERMES_ONE, initialize it to be constant 1.0.
  if(coeff == HERMES_ONE)
```

```
    this->coeff = new Hermes2DFunction<std::complex<double> >
    (std::complex<double>(1.0, 1.0));
}
```

In this constructor:

```
template<> DefaultMatrixFormVol<std::complex<double> >
// means that this is an explicit instantiation of a template for
// complex numbers (DefaultMatrixFormVol, the derived class, is actually also
// a template, as the prent class is).

int i, j
// coordinates in the system of equations, first is the row (basis functions),
// second the column (test functions).

std::string area //(typically optional)
// either a std::string for the marker on which this form will be evaluated,
// or HERMES_ANY constant for 'any', i.e. all markers
// (this is the default in the parent class constructor).

Hermes2DFunction<std::complex<double> >*coeff //(typically optional)
// custom function having its meaning specified in the
// calculating methods (see further). The constant HERMES_ONE,
// that really represents the number 1.0, is the default
// in the parent class constructor.

SymFlag sym //(typically optional)
// symmetry flag
// see the 'hermes-tutorial' documentation, section 'A-linear', chapter '03-poisson'.

GeomType gt //(typically optional)
// type of geometry: HERMES_PLANAR, HERMES_AXISYM_X, HERMES_AXISYM_Z,
// to distinguish between the normal 2D settings (HERMES_PLANAR),
// or an axisymmetric one. See the 'hermes-tutorial' documentation,
// section 'A-linear', chapter '09-axisym' for more details.
```

In those, the main methods to override are value(...), and ord(...), calculating the value and integration order respectively. It is a good idea to refer to the default forms (located in the library repository, with headers in hermes2d/include/weakform_library/*.h and the sources in hermes2d/src/weakform_library/*.cpp). The header is pretty self-explanatory:

```
// MatrixFormVol - MatrixFormSurf differs in the use of GeomSurf instead of GeomVol.
virtual Scalar value(int n, double *wt, Func<Scalar> *u_ext[], Func<double> *u,
  Func<double> *v, GeomVol<double> *e, Func<Scalar> **ext) const;

// A typical implementation.
template<typename Scalar> Scalar DefaultMatrixFormVol<Scalar>::value(int n,
  double *wt, Func<Scalar> *u_ext[], Func<double> *u, Func<double> *v,
  GeomVol<double> *e, Func<Scalar> **ext) const
{
  Scalar result = 0;

  for (int i = 0; i < n; i++)
    result += wt[i] * coeff->value(e->x[i], e->y[i]) * u->val[i] * v->val[i];
}

// VectorFormVol - VectorFormSurf differs in the use of GeomSurf instead of GeomVol.
// Identical to MatrixFormVol, only the basis function is missing for obvious reasons.
virtual Scalar value(int n, double *wt, Func<Scalar> *u_ext[], Func<double> *v,
    GeomVol<double> *e, Func<Scalar> **ext) const;
```

In these:

```
int n
// number of integration points.

double *wt
// integration weights (an array containing 'n' values).

Func<Scalar> *u_ext
// values from previous Newton iterations, as many as there are spaces
// (equations) in the system.

Func<double> *u
// the basis function, represented by the class Func.
// For more info about the class, see the developers documentation (in doxygen).
// How to get that, see the documentation section.

Func<double> *v
// the test function, represented by the class Func.
// For more info about the class, see the developers documentation (in doxygen).

GeomVol<double> *e
// geometry attributes: coordinates, element size,
// normal directions (for surface forms), you name it.
// - this is for volumetric forms (for surface forms one uses GeomSurf).
// For more info about the class, see the developers documentation (in doxygen).

Func<Scalar> **ext
// external functions, as many as you like
// (provided you set it up in constructor of your weak formulation
// derived from the class WeakForm).
// For more info about the class, see the developers documentation (in doxygen).
```

Now we have a space and a weak formulation, we are ready to calculate!

4.2.4 Calculation

We are going to get a solution vector from what we already have in the most general setup. This means for a time-dependent, adaptive example. This is to illustrate the various classes and methods, and the best thing about them, they are used pretty much the same way.

For details about time-dependent examples, and various aspects of that, see the 'hermes-tutorial' documentation, section 'C-transient'. For details about adaptive examples, and various aspects of that, see the 'hermes-tutorial' documentation, section 'D-adaptivity'. Right here we focus on the calculation:

```
double current_time = 'some number';
double current_time_step = 'also some number';
Time-loop
{
  Adaptive-loop // not necessarily on each time step.
  {
    // create reference space(s), see the adaptivity section of hermes-tutorial
    // documentation for this. e.g.
    Space<double>* ref_space = construct_refined_space(&space);

    // WE ARE NOW HERE.
    // The calculation
    // WE ARE NOW HERE

    // do the adaptivity thing, see the adaptivity section of hermes-tutorial
    // documentation for this.
    // This would change the 'coarse' Space instance: 'Space<double> space'.

    // Do some cleaning.
```

```
  }
  // adjust time, and time step any way you want
  // (stability conditions, time-adaptivity, ...).
}
```

In the following, basically everywhere where one can pass an instance of Space<Scalar>*, one can pass an instance of std::vector<Space<Scalar>*>, if the problem is a system of equations.

We shall start from the simplest case.

4.2.5 1 - linear example

Once we created the Space(s), and the WeakForm (always one!), we create (outside of the loops!) an instance of LinearSolver<Scalar>:

```
// Let us say that for real numbers, but for complex, it would be analogic.
LinearSolver<double> solver(&space, &wf).
```

Then the following code inside the loops would do the trick:

```
solver.set_time(current_time);
solver.set_time_tep(current_time_step);
// Yes! You are right, these can be used outside of the adaptivity loop!

// Set the new Space.
solver.set_space(ref_space);

// This is usually the place where something can go wrong,
// so we use the try-catch block. Note that the exceptions
// we use in Hermes are std::exception descendants (so only one catch block is enough),
// but you can choose to act differently upon a different exception type as shown.
try
{
  // Do the magic (assemble the matrix, right-hand side, solve).
  solver.solve();
}
catch(Hermes::Exception::Exception& e)
{
  e.print_msg();
}
catch(std::exception& e)
{
  std::cout << e.what();
}

// Get the solution.
// The Solution class is described in the developers (doxygen) documentation.
// The method vector_to_solution(s) too.
MeshFunctionSharedPtr<double> ref_sln(new Solution<double>);
Solution<double>::vector_to_solution(solver.get_sln_vector(), ref_space, ref_sln);
```

And that is it, we have the solution of the problem in that adaptivity step on that time level. What to do with it (visualize, do some calculations, projections, limiting, whatever) and how to do it is described in various points in the tutorial. But let us say that we would like to see it, the following will make us happy:

```
ScalarView<double> view("My Solution");
view.show(ref_sln);
```

4.2.6 2 - nonlinear example

In this surprisingly short section, we will learn how to use NewtonSolver, and PicardSolver.

It is literaly the same as in the previous section, just take out 'LinearSolver', and pass 'NewtonSolver', or 'PicardSolver'.

There is one more thing, if you want your NewtonSolver not to start from a zero initial guess, the following helps:

```
// Initialize the vector for initial guess. Real case.
// Also do not forget to 'delete []' this after you do not need it.
double* coeff_vec = new double[Space<double>::get_num_dofs(&ref_space)];

// For example let us project the previous time level solution and
// use it as an initial guess.
// And now use it in the NewtonSolver<Scalar>::solve.
// (solver is now NewtonSolver<double>) method.
solver.solve(previous_time_level_sln);
```

One can also use the NOX solver from the Trilinos package (with analogic, but not exactly same methods). One needs Trilinos for that. And documentation for that is coming.

4.2.7 3 - RungeKutta solver.

Again, pretty much the same as in the LinearSolver case, but the solve() method will now take the previous time level Solution(s) and return the new Solution(s), so there is no need for using the vector_to_solution(s) method:

```
// Initialize the solution(it can be outside of the loops,
// the solution would always be rewritten when it is natural)
MeshFunctionSharedPtr<double> ref_sln(new Solution<double>);

// "solver" is now an instance of RungeKutta<double>.
solver.set_time(current_time);
solver.set_time_tep(current_time_step);

// Yes! You are right, these can be used outside of the adaptivity loop!
// Set the new Space.
solver.set_space(ref_space);

// This is usually the place where something can go wrong,
// so we use the try-catch block. Note that the exceptions
// we use in Hermes are std::exception descendants (so only one catch block is enough).
try
{
  // Do the usual magic, plus put the result in the ref_sln instance.
  solver.solve(previous_time_level_sln, ref_sln);
}
catch(std::exception& e)
{
  std::cout << e.what();
}
```

4.2.8 4 - use DiscreteProblem class directly

For special purposes, like DG or FVM (Finite Volume Method), where one needs to access the matrix or right-hand side, or needs to have the solution in hand before projection (to do limiting etc.), one can also directly use this class.

It shares some methods with the above 'calculation' classes, but of course does not do any calculation. The usage would look like this:

```
// We assume we have an instance DiscreteProblem<double> dp(&wf, &space);

// These can be outside the loop, the memory would get properly freed / reallocated
// every time without worrying about it.
```

```
SparseMatrix<double>* matrix = create_matrix<double>();
Vector<double>* rhs = create_vector<double>();
LinearMatrixSolver<double>* linear_matrix_solver = create_linear_solver<double>(matrix, rhs);

dp.set_time(current_time);
dp.set_time_tep(current_time_step);

// Set the new Space.
dp.set_space(ref_space);

// This is usually the place where something can go wrong,
// so we use the try-catch block. Note that the exceptions
// we use in Hermes are std::exception descendants (so only one catch block is enough).
try
{
  dp.assemble(matrix, rhs);

  // NOW WE HAVE THE MATRIX and RHS ASSEMBLED and we can do whatever we want with it.
  linear_matrix_solver.solve();
}
catch(std::exception& e)
{
  std::cout << e.what();
}

// Get the solution.
// The Solution class is described in the developers documentation.
// The method vector_to_solution(s) too.
MeshFunctionSharedPtr<double> ref_sln(new Solution<double>);
Solution<double>::vector_to_solution(linear_matrix_solver.get_sln_vector(), ref_space, ref_sln);
```

And that is it. There is not much more to it. See the 'transient', and 'adaptivity' sections of the hermes-tutorial documentation and all will fall into place.

Of course every problem is different, such as in the case of DG, one needs to do some limiting, shock capturing etc... One can also save / load various entities (Spaces, Solutions, Meshes, time steps, ...) during calculation.

And especially, one needs to be careful not to forget deallocating stuff. How to do that, see the hermes-tutorial, and hermes-examples repositories. The examples there should be done properly.

4.3 Introduction to advanced C++ object-oriented features

There is plenty of material on the internet that is concerned with advanced C++.

This list was put together in February, 2013

- http://www.cs.utexas.edu/~jbsartor/cs105/CS105_spr10_lec8.pptx.pdf
- http://www.cs.ust.hk/~dekai/library/ECKEL_Bruce
- http://www10.informatik.uni-erlangen.de/~pflaum/pflaum/ProSeminar/meta-art.html

4.4 Hermes C++ object model - deriving your own specialized classes

DOXYGEN documentation

Anything that is written here can be much faster and in much more complex way read and understood from the Doxygen documentation - please, please, use it - many an hour has been spent to make Doxygen documentation a useful resource for programmers.

There are several classes that represent some piece of the whole FEM discretization and calculation process that hold the custom informaion for a specific problem.

Be sure to have checked the section "Typical example" that will give you an idea of the basics. This section is more technically focused.

These are

- Forms (= integrals) of the weak formulation
- Essential (= Dirichlet) boundary conditions
- Mesh functions (= initial conditions, exact solutions, etc.)
- Mathematical functions (= nonlinear relations, etc.)
- Filters (= functions of solutions, for post-processing)

Each of these is described in what follows.

4.4.1 Notes on templating

In the section of advanced Object-Oriented aspects of C++ and especially its template metaprogramming capabilities, you should find enough information on the subject to start with, should you need. The templates in Hermes are very simple. Most classes (and all that are discussed in this section) come only in two forms - there exist only two *template instantiations* - for real and for complex numbers. Also, since we do not see much use in using single-precision calculations, only the two following instantiations exist for most classes:

- double
- std::complex<double>

So whenever there is a class header that looks like this:

```
template<typename Scalar>
class AClass
{
  ...
};
```

It is exactly this case, you can use AClass<double> for real problems, and AClass<std::complex<double> > for complex problems.

The explicit instantiations, should you need to add such, need to be at the end of the **source** file (.cpp), **not** the header file.

4.4.2 Forms

Weak formulation is the key part of the FEM discretization. The whole weak formulation with all the integrals that it contains and their specification (over what part of the domain is the integration, on which coordinates of the system of PDEs it is located, etc.) is represented by the class template

WeakForm<Scalar> (file: hermes2d/include/weakform/weakform.h)

In the implementation of your programs, you need to subclass (or derive from) this class as was seen in the "Weak formulation" part of the "Typical example" section:

class MyWeakForm : public WeakForm<double>

What is necessary to say is that WeakForm is usually passed by means of shared pointers, using class template WeakFormSharedPtr (see examples for how to use this).

The important methods and attributes to note in this class are:

- the constructor WeakForm(unsigned int neq = 1, bool mat_free = false)

  - neq stands for Number Of Equations
  - in the body, you need to use the following methods to add single forms (integrals) to the weak formulation
    * void add_matrix_form(MatrixFormVol<Scalar>* mfv);
    * void add_matrix_form_surf(MatrixFormSurf<Scalar>* mfs);
    * ...
  - other methods (set_ext(...), set_current_time(...)) are described in the Doxygen documentation.
  - **clone()** - the cloning method, for parallelization
    * the purpose of this method is to create an exact copy of the instance at hand (as simple as that - return a copy, but with no shared data)
    * the reason is that in the openMP parallel assembling, each thread will receive a copy of the instance for its own processing (and this will eliminate the potential parallelization issues)
    * this is there just for the case that you have any data that are not thread-safe and you do not want to take care about this on your own (using openMP directives like **pragma omp critical** etc.)

Right after the WeakForm come the most difficutlt and problem causing classes - **Forms** - here is the list of form types:

- MatrixFormVol - a bilinear form that represent integration over a volume
- MatrixFormSurf - a bilinear form that represent integration over a surface
- MatrixFormDG - a bilinear form that represent integration over an inner edge - for DG
- VectorFormVol - a linear form that represent integration over a volume
- VectorFormSurf - a linear form that represent integration over a surface
- VectorFormDG - a linear form that represent integration over an inner edge - for DG

How to subclass these should be obvious from the "Weak formulation" part of the "Typical example" section.

4.4.3 Essential boundary conditions

There is three classes to note for handling of Dirichlet boundary conditions (all in hermes2d/include/boundary_conditions/essential_boundary_conditions.h)

- DefaultEssentialBCConst - a class you may use directly, without any subclassing
  - only thing is to specify the constant value in the constructor:

    DefaultEssentialBCConst(std::string marker, Scalar **value_const**);
- DefaultEssentialBCNonConst - a class you again may use directly, **or** subclass, should you wish to use a non-Constant Dirichlet BC
  - if you wish to use the class directly, you need to specify an instance of the following in the constructor:

    ExactSolutionScalar<Scalar>
  - if you wish to use a specific value function, you may just subclass this class and override the value() method
  - instructions for subclassing can be found in the file hermes2d/include/boundary_conditions/essential_boundary_conditions.h just above the class
- EssentialBCs - a class you will never subclass, it is a container that you pass to a constructor of a Space (see the "Space" part of the "Typical example" section).

An example usage of the non-constant boundary condition with subclassing is in the test examples:

hermes2d\test_examples\03-navier-stokes\definitions.cpp at the very bottom

4.4.4 Mesh functions

Very important thing regarding the Mesh functions is the use of shared pointers, in this sense, the class template MeshFunctionSharedPtr, in this example the mesh function is a solution:

```
MeshFunctionSharedPtr<double> my_mesh_function(new Solution<double>(mesh));
```

An example of this is the following code from the test example 06:

hermes2d\test_examples\06-system-adapt\definitions.cpp (.h)

The point here are the two classes ExactSolutionFitzHughNagumo1, ExactSolutionFitzHughNagumo2. This is the definition of the class **ExactSolutionFitzHughNagumo1** and the declaration of its methods in definitions.h:

```
class ExactSolutionFitzHughNagumo1 : public ExactSolutionScalar<double>
{
public:
  ExactSolutionFitzHughNagumo1(MeshSharedPtr mesh);

  virtual double value(double x, double y) const;

  virtual void derivatives(double x, double y, double& dx, double& dy) const;

  virtual Ord ord(double x, double y) const;

  ~ExactSolutionFitzHughNagumo1();

  virtual MeshFunction<double>* clone() const;

  CustomExactFunction1* cef1;
};
```

Note the subclassing line:

```
// This should be obvious for any C++ user
class ExactSolutionFitzHughNagumo1 : public ExactSolutionScalar<double>
```

Then we can see the important methods are overriden in the source file definitions.cpp:

- value(double x, double y) const

```
double ExactSolutionFitzHughNagumo1::value(double x, double y) const
{
  return cef1->val(x)*cef1->val(y);
}
```

- derivatives(double x, double y, double& dx, double& dy) const

```
void ExactSolutionFitzHughNagumo1::derivatives(double x, double y,
  double& dx, double& dy) const
{
  dx = cef1->dx(x)*cef1->val(y);
  dy = cef1->val(x)*cef1->dx(y);
}
```

- ord(double x, double y) const

```
Ord ExactSolutionFitzHughNagumo1::ord(double x, double y) const
{
  return Ord(10);
}
```

- clone() const

```
MeshFunction<double>* ExactSolutionFitzHughNagumo1::clone() const
{
  return new ExactSolutionFitzHughNagumo1(this->mesh);
}
```

4.4.5 Mathematical functions

Once again we shall use the example 06:

```
hermes2d\test_examples\06-system-adapt\definitions.cpp (.h)
```

In the header file (definitions.h) we can see the following class definition:

```
class CustomRightHandSide1: public Hermes2DFunction<double>
{
public:
  CustomRightHandSide1(double K, double d_u, double sigma);

  virtual double value(double x, double y) const;

  virtual Ord value(Ord x, Ord y) const;

  ~CustomRightHandSide1();

  CustomExactFunction1* cef1;
  CustomExactFunction2* cef2;
  double d_u, sigma;
};
```

The important methods here are (definitions - method bodies from definitions.cpp):

- value(double x, double y) const

```
double CustomRightHandSide1::value(double x, double y) const
{
  double Laplace_u = cef1->ddxx(x) * cef1->val(y)
                   + cef1->val(x) * cef1->ddxx(y);
  double u = cef1->val(x) * cef1->val(y);
  double v = cef2->val(x) * cef2->val(y);
  return -d_u * d_u * Laplace_u - u + sigma * v;
}
```

- value(Ord x, Ord y) const

```
// Note here that we are saying that this function is ok to be integrated with a
// quadrature rule precise for polynomials of order 10.
Ord CustomRightHandSide1::value(Ord x, Ord y) const
{
  return Ord(10);
}
```

- Note that there is no **clone** method here. That is because these classes - mathematical functions - are used in OpenMP paralell blocks only inside methods of already cloned class instances - like Form::value() etc.

4.4.6 Filters

There is a number of pre-defined Filters for you in:

`hermes2d\include\function\filter.h`

These include

- AngleFilter
- VonMisesFilter
- LinearFilter
- ValFilter
- MagFilter
- ...

They all come from the base class template Filter<Scalar>.

Underneath there is a distinction between the filters that come from the classes SimpleFilter or DXDYFilter (real function of real solutions, or complex one of complex) and those coming from ComplexFilter (real function of complex solutions).

The difference is obvious, the Solution<Scalar> template instances it operates with differ: for SimpleFilter / DXDYFilter successors, the type (real vs. complex) depends on the type of the filter, and in the case of ComplexFilter successors it is always:

```
MeshFunction<std::complex<double> >* solution
```

Also note that whereas SimpleFilter / DXDYFilter are class **templates** - as explained in the previous paragraph, the ComplexFilter is just a class, and it inherits from Filter<double>.

SimpleFilter serves for functions of the solutions(s) values, DXDYFilter for functions of the solution(s) derivatives.

The common method all filters must override is:

```
virtual MeshFunction<Scalar>* clone() const
```

Then there is always the method **filter_fn(...)** that comes in the following versions:

```
// SimpleFilter - values here represent the solution values, n is the number of points.
virtual void filter_fn(int n, const std::vector<const Scalar*>& values, Scalar* result) = 0;

// DXDYFilter - contains values, dx - derivatives w.r.t. x, dy - derivatives w.r.t. y,
// and also the resulting derivatives, should those be necessary.
virtual void filter_fn (int n, const std::vector<Scalar *>& values, const std::vector<Scalar *>& d

// ComplexFilter - values here represent the solution values, n is the number of points,
// note that here, the values are complex.
virtual void filter_fn(int n, const std::complex<double>* values, double* result) = 0;
```

CHAPTER

FIVE

EXTENDED DOCUMENTATION

5.1 Hermes Documentation overview

5.1.1 Building user documentation (this one) in HTML

Before building User Documentation, install the Python Sphinx package:

```
Linux: sudo apt-get install python-sphinx
Windows: download Python (http://python.org), download setup tools (http://pypi.python.org/pypi/s
```

The user documentation can be found in the directory doc/. Type "make html" there to build it. The HTML pages can then be displayed by typing:

```
firefox _build/html
```

or using another web browser.

5.1.2 Building user documentation (this one) in PDF

In the directory doc/ type "make latex" and you will be instructed how to build the PDF:

```
Build finished; the LaTeX files are in _build/latex.
Run 'make all-pdf' or 'make all-ps' in that directory to run these through (pdf)latex.
```

5.1.3 Developer Documentation (in Doxygen)

The documentation is accessible online.

> Hermes - common code.
>
> Hermes - 2D specific code.

In order to build developers documentation, install Doxygen:

```
Linux: sudo apt-get install doxygen
Windows: `<http://ftp.stack.nl/pub/users/dimitri/doxygen-1.8.2-setup.exe>`_.
```

There are separate Doxygen files for hermes-common (dimension-independent functionality such as matrix solvers) and for Hermes2D. To build the former, go to the directory hermes-common/ and type "doxygen Doxyfile". The HTML docs will be located in a new subfolder doc/html/ and you can view them in any web browser, such as:

```
firefox doc/html/index.html
```

To build Doxygen files for Hermes2D, go to the directory hermes2d/ and type again "doxygen Doxyfile". The HTML docs will be located in the file doc/html/index.html.

5.1.4 Other Resources

See the section "Citing Hermes" in this document for a representative selection of books and scientific papers about Hermes. A more complete overview of publications about Hermes and adaptive *hp*-FEM can be found in the publications section. at hpfem.org. The most recent list is probably the one on Pavel Solin's publications page.

5.2 Hermes Tutorial

5.2.1 Getting the repository with tutorial code

To get the hermes-tutorial repository, clone:

```
git clone git://github.com/hpfem/hermes-tutorial.git
```

After cloning, go to hermes-tutorial/doc folder and type "make" in the doc/ directory.

Please refer to the documentation there for the instructions how to install the tutorial in the hermes-tutorial repository and for the step-by-step tutorials descriptions.

5.3 Examples Documentation

5.3.1 Getting the repository with examples code

To get the hermes-examples repository, clone:

```
git clone git://github.com/hpfem/hermes-examples.git
```

After cloning, go to hermes-examples/doc folder and type "make" in the doc/ directory.

Please refer to the documentation there for the instructions how to install the examples in the hermes-examples repository and for the step-by-step examples descriptions.

CHAPTER

SIX

COLLABORATION

6.1 Collaboration via Github

The following is a very simple primer on Github and Git whose objective is to show you how to effortlessly download Hermes and eventually contribute to the project by creating an interesting example, fixing a bug, improving documentation, etc. We begin with creating a free account at Github, fork the Hermes git repository, clone your Github copy to your local computer, and continue through setting up the .gitconfig file, creating a new branch, committing changes, pushing them to your Github repository, and generating a pull request. Some good references for further reading are given after that.

6.1.1 Create a Free Github Account

Go to the Github home page. Click on "Plans, Pricing and Signup" and then on "Create a free account". You'll be asked to enter your username, email, and a password. That's it, it does not take much time, and it is free for open source projects.

6.1.2 Fork the Hermes Git Repository

Enter your account using the "Login" link in the upper right corner. Type "hpfem" into the search line in the upper right corner and hit enter. You will see a list of several repositories and one of them will be "hpfem / hermes". Click on it and this will bring you to the Hermes page at Github. Click on the "Fork" button in the upper part of the page. This will create a copy of the Hermes repository in your Github account, establish important links to the master Hermes repository, and add you to the Hermes network.

6.1.3 Generate and Register Your Public SSH Key

Start by generating the public ssh key typing (on Linux):

```
ssh-keygen
```

You will be asked to enter a file for the key to be stored, and a passphrase. The passphrase may be left empty. After the file with the key is generated, open it in some text editor. Then return to your Github page, click on "Account Settings", and on "SSH Public Keys". This is where you need to paste the public key that you just generated on your computer.

6.1.4 Download Hermes to Your Local Computer

Once your public SSH key is registered at Github, you can clone the Hermes repository from your Github account to your local computer. This is done by typing:

```
git clone git@github.com:your_name/hermes.git
```

Now you can build Hermes as described in the Installation section above.

In the next paragraphs we describe how to work inside the Hermes Git repository on your local computer.

6.1.5 Create the .gitconfig File

The .gitconfig file can be used to define your identity for git as well as useful abbreviations. Change dir to your home directory. Then adjust and save the following as "~/.gitconfig":

```
[user]
        name = Pavel Solin
        email = solin.pavel@gmail.com

[core]
        editor = vim

[color]
        ui = true
[color "branch"]
        current = yellow reverse
        local = yellow
        remote = green
[color "diff"]
        meta = yellow bold
        frag = magenta bold
        old = red bold
        new = green bold
        whitespace = red reverse
[color "status"]
        added = yellow
        changed = green
        untracked = cyan
[core]
        whitespace=fix,-indent-with-non-tab,trailing-space,cr-at-eol

[alias]
        st = status
        ci = commit
        br = branch
        co = checkout
        df = diff
        lg = log -p
```

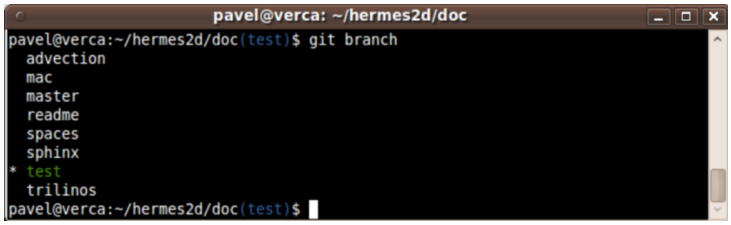
6.1.6 Create a Local Branch

Change dir back to hermes/hermes2d/ or hermes/hermes3d/ where you were before. Whenever you want to do any changes, such as modify an existing example or create a new one, always create a local branch - do not work in the master branch.

You can get an overview of existing branches by typing:

```
git branch
```

This will show you something like this:



If this is your first time, then you will see just the master branch with the star next to it, which tells you that there are no other branches.

A new branch is created by typing:

```
git co -b test-1
```

where test-1 is the name of your new local branch. Now you can do any changes you like and you do not have to be afraid of damaging your master branch. HOWEVER, you always must commit your changes as described below. Unless you commit your changes, Git does not know that they belong to your local branch. This may cause conflicts, you may not be able to update your local repository, you may not be able to switch branches at all, etc.

6.1.7 Commit Your Changes

Say that you modified an existing or added a new file "file.cpp". In order to register the new changes, type:

```
git add file.cpp
```

You can do this with as many files as you like. Use the command:

```
git diff
```

to see whether you have unregistered changes. If all your changes are registered, the command will print nothing.

After all your changes are registered, type:

```
git commit
```

This will invoke a basic text editor where you will be asked to enter a one-line comment describing your changes. Without this line, your commit will not be accepted.

6.1.8 Push the Changes to Your Github Account

You cannot push to the master repository of Hermes directly. The way to get your changes there is to first push them to your Github fork and then send a pull request to the Hermes network. To push your changes, type:

```
git push git@github.com:your_name/hermes.git test-1:test-1
```

This will push your local branch test-1 to a branch of the same name at Github. Now you can go back to your Github account, click on "Commits" and you should see your changes there.

6.1.9 Send a Pull Request

In order to submit your changes to the Hermes network, click on the button "Pull request" in the upper right part of your Github page. Describe the changes you did in the text window that appears. On the right you can see a list of people who wil be notified about your changes. You can preview your pull request by clicking on "Preview" above the text window. The source and target branch are displayed above the text window and you can change them when you click on them. When you are ready, click on "Send pull request".

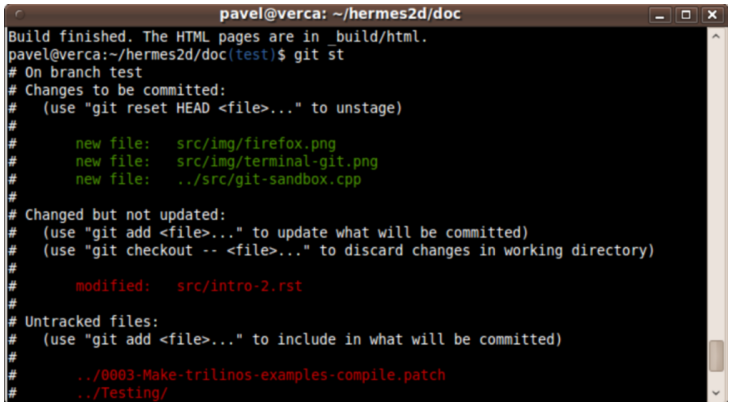
For more details on pull requests visit this page.

6.1.10 Switching Branches

Before changing to a different branch in your local repository, type:

```
git status
```

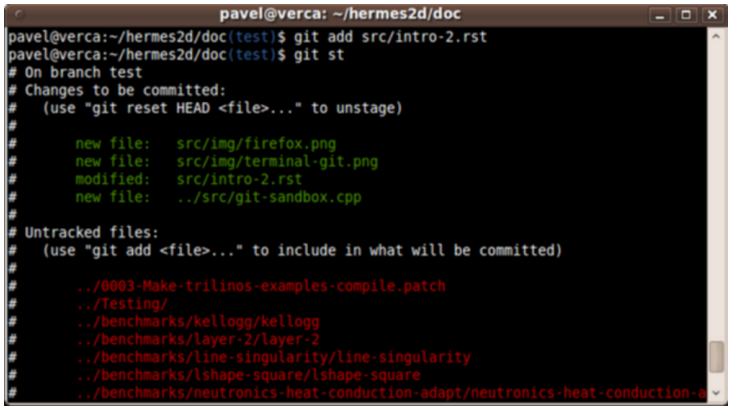
You will see something like this:



The green font tells you that git has the latest version of the file. All modified files in red need to be added using "git add". It is a good idea to go through the untracked files too, in case that you wish to add some of them as well. Related to the sample screenshot above, after typing:

```
git add src/intro-2.rst
git st
```

you will see



Now you can proceed with "git commit" as described above. After the commit, you can switch to a different branch by typing:

```
git co branch-name
```

6.1.11 Further Reading

Git and Github are very powerful tools and we covered just a tiny part of the story. After you familiarize yourself with the contents of this simple primer, read more in Pro Git.

Also watch this YouTube video. by Scott Chacon.

Good luck and let us know if you think that this document could be improved!