FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Hermes library

Rapid hp-FEM & hp-DG Solver Toolkit

Lukas Korous et al.

hp-FEM group, University of West Bohemia, Pilsen; University of Nevada, Reno

August 3, 2014

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

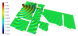
FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

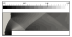
When Hermes comes in handy

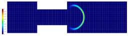
- Algorithms testing, methods comparison
  - D. Pugal, P. Solin, J. K. Kim, A. Aabloo: Modeling Ionic Polymer-Metal Composites with Space-Time Adaptive Multimesh hp-FEM, Communications in Computational Physics 2012, 11 (249-270)
  - M. Bittl, D.Kuzmin: An hp-adaptive flux-corrected transport algorithm for continuous finite elements, Computing, May 2013, 95-1 Supplement (27-48)
- Adaptive algorithms benchmarking
  - P. Solin, O. Certik, L.Korous: Three anisotropic benchmark problems for adaptive finite element methods, Applied Mathematics and Computation, 2013, 219-13 (7286-7295)
  - Z. Ma, L. Korous, E. Santiago: Solving a suite of NIST benchmark problems for adaptive FEM with the Hermes library, Journal of Computational and Applied Mathematics, 2012, 236-18 (4846-4861)
- Easy solver prototyping
  - Acoustics, Wave propagation
  - Compressible & Incompressible Flow
  - Flame Propagation
  - Maxwell’ss equations (transient, harmonic)
  - Linear Elasticity, Thermoelasticity
  - Gross-Pitaevski, Nernst-Planck
  - Neutronics (Milan Hanus - NTIS, UWB)
  - Richards Equation
  - ...

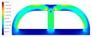
When Hermes comes in handy

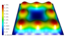
- Acoustics, Wave propagation
- Compressible & Incompressible Flow
- Flame Propagation
- Maxwell'ss equations (transient, harmonic)
- Linear Elasticity, Thermoelasticity
- Gross-Pitaevski
- Neutronics (Milan Hanus - NTIS, UWB)
- Richards Equation

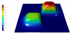


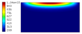












FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Easy installation

Industry standard tools

- **Git**, github.com
- **CMake** multiplatform build system
- Code compliant with **g++**, **MSVC** 2010 and newer
- **Doxygen** documentation available online
- **XML** exportable & importable entities
- **OpenGL** visualization & VTK, Tecplot outputs
- No Fortran - only **C++**

Easy installation

Linux

Primary supported distribution is Ubuntu, Debian. The following is the complete procedure how to install Hermes

1. sudo apt-get install git git-core cmake g++ freeglut3-dev libsuitesparse-dev libglew-dev libxerces-c-dev xsdcxx
2. git clone git@github.com:hpfem/hermes.git
3. cd hermes
4. cmake .
5. make -j#
6. sudo make install

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Easy installation

Windows

Microsoft Visual Studio is the primary supported compiler, IDE and debugger on Windows. The following is the complete procedure how to build Hermes

1. Download prerequisities from http://www.hpfem.org/hermes/building-and-using-hermes-on-windows/
2. Download Git client (git-scm.com/) and CMake (cmake.org/)
3. git clone git@github.com:hpfem/hermes.git
4. cd hermes
5. cmake -G Visual Studio #
6. devenv.exe hermes.sln
7. Build

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Learning resources

- **Library documentation**
  - **HTML** - http://hpfem.org/hermes-doc/hermes/html/index.html
  - **PDF** - http://hpfem.org/hermes-doc/hermes/hermes.pdf
- **Tutorial & Examples**
  - **Tutorial** - http://hpfem.org/hermes-doc/hermes-tutorial/html/index.html
  - **Examples** - http://hpfem.org/hermes-doc/hermes-examples/html/index.html
- **Doxygen documentation**
  - **Common** (dimension independent) tools:

    http://hpfem.org/hermes-doc/hermes/hermes_common/index.php
  - **Hermes2D** http://hpfem.org/hermes-doc/hermes/hermes2d/index.php

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Coding with Hermes

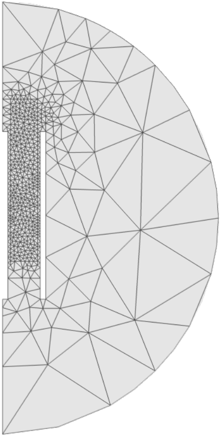
Mesh

```xml
<?xml xversion="1.0" encoding="UTF-8" stand
<domain:domain xmlns:domain="XMLSubdomains"
 xmlns:xsi="http://www.w3.org/2001/XMLSchem
xsi:schemaLocation="XMLSubdomains
C:/hpfem/hermes/resources/xsd/subdomains_h2
C:/hpfem/hermes/resources/xsd/mesh_h2d_xml.
  <vertices>
    <v i="0" x="-9.22506e-17" y="0.2"/>
    <v i="1" x="0" y="0.08"/>
    <v i="2" x="0.005" y="0.08"/>
    <v i="3" x="0.035" y="0.08"/>
    <v i="4" x="0.04" y="0.08"/>
    <v i="5" x="1.5022e-17" y="-0.2"/>
```

(a) XML mesh format

```cpp
using namespace Hermes::Hermes2D;

// Load the mesh.
MeshSharedPtr mesh(new Mesh);
MeshReaderH2DXML mloader;
mloader.load("actuator.xml", mesh);

// Initial mesh refinements.
mesh->refine_towards_boundary(BDY_OBSTACLE, 2, false);
mesh->refine_towards_boundary(BDY_TOP, 2, true);
mesh->refine_all_elements();
```

(c) Hermes2D classes to handle mesh



(b) Mesh visualized using OpenGL

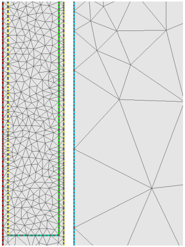
FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Coding with Hermes

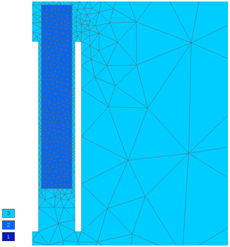
Space, boundary conditions

```
using namespace Hermes::Hermes2D;

// Initialize boundary conditions.
EssentialBCNonConst bc_left_vel_x(BDY_LEFT, VEL_INLET, H, STARTUP_TIME);
DefaultEssentialBCConst<double> bc_other_vel_x(Hermes::vector<std::string>(BDY_BOTTOM,
EssentialBCs<double> bcs_vel_x(Hermes::vector<EssentialBoundaryCondition<double> *>(&bc
DefaultEssentialBCConst<double> bc_vel_y(Hermes::vector<std::string>(BDY_LEFT, BDY_BOTT
EssentialBCs<double> bcs_vel_y(&bc_vel_y);
EssentialBCs<double> bcs_pressure;

// Spaces for velocity components and pressure.
SpaceSharedPtr<double> xvel_space(new H1Space<double>(mesh, &bcs_vel_x, P_INIT_VEL));
SpaceSharedPtr<double> yvel_space(new H1Space<double>(mesh, &bcs_vel_y, P_INIT_VEL));
SpaceSharedPtr<double> p_space(new L2Space<double>(mesh, P_INIT_PRESSURE));
```

(a) BCs and Space handling in Hermes2D



(b) Boundary markers



(c) Space - polynomial order

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Coding with Hermes

Weak formulation

$$\boxed{\frac{\boldsymbol{v}^{n+1}}{\tau}} - \frac{\boldsymbol{v}^{n}}{\tau} - \boxed{\frac{1}{Re}\Delta\boldsymbol{v}^{n+1}} + \boxed{(\boldsymbol{v}^{n+1}\cdot\nabla)\boldsymbol{v}^{n+1}} + \nabla p^{n+1} = 0$$

$$\mathrm{div}\boldsymbol{v}^{n+1} = 0$$

(a) Navier-Stokes equations (strong formulation)

```
double value(int n, double *wt, Func<double> *prev_nonlin_sln[], Func<double>* v, Func<double> *v_test,
  Geom<double> *geom, Func<double> **prev_time_sln) const
{
  double result = 0;

  Func<double> *vx_prev = prev_time_sln[0], *vy_prev = prev_time_sln[1];

  for (unsigned short i = 0; i < n; i++)
  {
    result += wt[i] * v->val[i] * v_test->val[i] / time_step;
    result += wt[i] * (v->dx[i] * v_test->dx[i] + v->dy[i] * v_test->dy[i]) / Re;
    result += wt[i] * (vx_prev->val[i] * v->dx[i] + vy_prev->val[i] * v->dy[i]) * v_test->val[i];
  }

  return result;
}
```

(b) Weak formulation in Hermes2D

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Coding with Hermes

Solvers - linear and nonlinear (and others)

Linear ones are easy:

```
template <typename Scalar>
class LinearSolver : public Hermes::Hermes2D::Solver<Scalar>
{
public:
  /// Constructor - solver of the problem described by the
  /// provided weak formulation 'wf' in the provided FE space 'space'.
  LinearSolver(WeakForm<Scalar>* wf, Space<Scalar>* space);

  /// Basic solve method - here the initial guess serves
  /// only as the initial guess for iterative solvers.
  virtual void solve(Scalar* initial_guess);

  /// Return the solution vector.
  Scalar* get_sln_vector();
};
```

Figure : Linear solver API

For nonlinear ones we need more weaponry:

```
template<typename Scalar>
class HERMES_API NewtonSolver : public Hermes::Hermes2D::Solver<Scalar>
{
public:
  ...

  /// Set the maximum number of iterations, thus co-determine when to
  void set_max_allowed_iterations(int max_allowed_iterations);

  /// Various types of stopping criteria.
  enum NonlinearConvergenceMeasurementType
  {
    ResidualNormRelativeToInitial, ResidualNormRatioToInitial,
    ResidualNormAbsolute, SolutionChangeRelative, ...
  };

  /// Set the residual norm tolerance for ending the Newton's loop.
  /// ...
  void set_tolerance(double tolerance, NonlinearConvergenceMeasurement
    bool serialize_with_AND);

  /// Sets minimum damping coefficient.
  void set_min_allowed_damping_coeff(double min_allowed_damping_coeff_

  /// Make the automatic damping start with this coefficient.
  void set_initial_auto_damping_coeff(double coeff);

  /// Set the ratio for the automatic damping.
  /// ...
  void set_auto_damping_ratio(double ratio);

  /// Set maximum number of steps (Newton iterations) that a jacobian
  /// it is deemed a 'successful' reusal with respect to the improveme
  /// ...
  void set_max_steps_with_reused_jacobian(unsigned int steps);

  /// Set the improvement factor for a jacobian reuse
  /// ...
  void set_sufficient_improvement_factor_jacobian(double ratio);

  /// ...
```

Figure : Newton solver API

Coding with Hermes

Adaptivity

- Hermes can do hp-adaptivity with or without a reference solution.
- The reference solution approach is probably the most general.
- A common sense implementation would
  1. create a reference space from the currently considered (coarse) space
  2. solve (using e.g. NewtonSolver) the problem in the fine space
  3. solve in the coarse space
  4. subtract & express the difference as a function (is this trivial and cheap?)
  5. use the difference as data to decide whether to split in h or in p (how?)
  6. refine the space as decided and continue from top (is this trivial and cheap?)

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Coding with Hermes

Adaptivity

- Our implementation does more or less the same thing
    1. create a reference space from the currently considered (coarse) space
    2. solve (using e.g. NewtonSolver) the problem in the fine space
    3. ~~solve in the coarse space~~ Perform an orthogonal projection of the solution from the fine to the coarse space (much cheaper).
    4. subtract & express the difference as a function (Is this trivial and cheap? Somehow.)
        - Because we have both the coarse and fine component in hand, we can calculate the difference locally (in parallel for each element separately).
        - Only for refined elements - we do not have to evaluate the difference for the non-refined elements.
        - Need to use the fine component quadrature and evaluate the coarse one in those points.
    5. use the difference as data to decide whether to split in *h* or in *p* (How? It depends (many considerations).)
        - What elements to refine? - we should not refine all of them, stop somewhere so that the mesh is refined where appropriate.
        - How to choose among many refinement candidates? - according to some balancing strategy between error reduction and increased problem size.
        - Hermes contains prebuilt strategies, candidate sets etc. + it is easy to implement custom ones.
    6. refine the space as decided and continue from top (Is this trivial and cheap? Yes.)

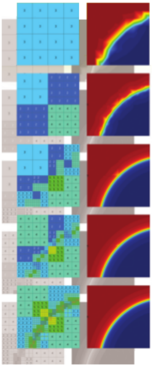
FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Coding with Hermes

Adaptivity

```
/// Default local error calculator in a specified norm.
DefaultErrorCalculator<double, HERMES_H1_NORM>
  errorCalculator(RelativeErrorToGlobalNorm, 1);

/// Stopping criterion for a particular adaptivity step.
AdaptStoppingCriterionSingleElement<double> stoppingCriterion(0.7);

/// Adaptivity processor class encapsulating the adaptivity functional
Adapt<double> adaptivity(space, &errorCalculator, &stoppingCriterion);

/// Predefined list of element refinement candidates.
const CandList CAND_LIST = H2D_HP_ANISO;

/// Projection based selector of refinement candidates
H1ProjBasedSelector<double> selector(CAND_LIST);

/// Stopping criterion for the adaptivity loop
/// (measured using errorCalculator).
const double ERROR_STOP = 5e-2;

/// Adaptivity loop.
do
{
  /// Solve...
  /// ... and project to obtain the coarse component:
  OGProjection<double>::project_global(space, solutionFine,
    solutionCoarse);

  /// Calculate the total error & element errors.
  double errorEstimate =
    errorCalculator.calculate_errors(solutionCoarse, solutionFine);

  /// We either have reached the threshold or we continue to adapt.
  adaptivityLoopDone = (errorEstimate < ERROR_STOP) ||
    adaptivity.adapt(&selector);

} while (!adaptivityLoopDone);
```

(a) Adaptivity API of Hermes



(b) hp-FEM solution of a benchmark

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

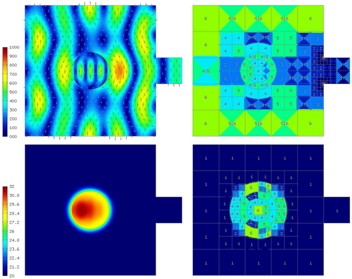
Features list

- Unified library for handling both **real and complex** problems using C++ templating
- Library capable of solving problems using **hp-FEM** and **hp-DG** methods and their **combination**
- Shared memory parallel code (using **OpenMP**) tuned for performance
- XML & binary **export / import** of important algebraic & FEM-related entities
- **Multimesh**: Calculations with physical quantities defined in different subdomains on separate meshes
- Comfortable development: Exception safe API, **OpenGL** visualization in separate thread
- Supported platforms: Linux (Ubuntu, Debian), Windows (MSVC)
- Well arranged doxygen & sphinx **documentation** with described examples
- Generic support (in all physical fields) of **curved elements**
- Shapesets of p-order up to 10 **+** base classes for **custom shapeset implementations**

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Features list

Microwave heating



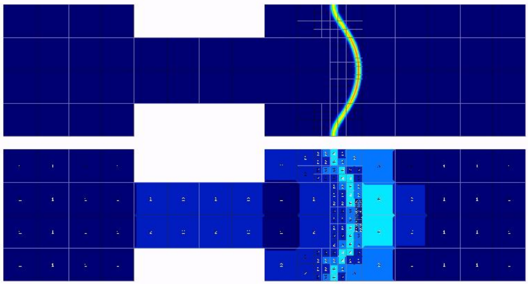
The microwave model consists of a cavity and a small square waveguide attached to its righ-hand side.

The cavity contains a food specimen (load) with temperature-dependent electric permittivity.

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Features list

Flame propagation

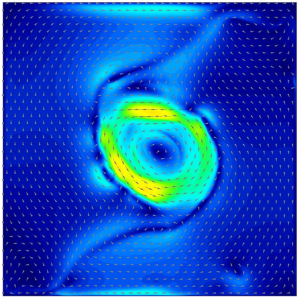


The underlying model consists of two nonlinear parabolic equations describing the temperature and concentration. The flame moves through the domain from the left to the right.

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Features list

Two-component incompressible viscous flow



The problem is described by Navier-Stokes equations for two-component flow with some modifications.

FACULTY OF ELECTRICAL ENGINEERING
UNIVERSITY OF WEST BOHEMIA

Live presentation

http://www.hpfem.org/hermes/