FEMLISP - a Common Lisp framework for solving partial differential equations

Nicolas Neuss
IANM, University Karlsruhe

Contents

- Partial Differential Equations
- The Finite Element Method
- Using Femlisp
- Why Common Lisp?
- Femlisp and others
- Discussion

Partial Differential Equations – 1

Observation: Phenomena which are characterized by

- instantaneous and short-range interactions
- in a continuum

can be modeled by partial differential equations (PDEs).

Examples: continuum mechanics, fluid mechanics, reaction and transport, general relativity, quantum mechanics

Applications: physics, chemistry, biology, economy, . . . , many engineering disciplines

Partial Differential Equations – 2

Definition: A PDE is an equation for an unknown *function* $u : \Omega \to \mathbb{R}$ which is satisfied for all points $x \in \Omega \in \mathbb{R}^n$ and involves only the function values and its derivatives at x.

Examples:

- 2D-Convection: $a(x,y)\frac{\partial}{\partial x}u(x,y) + b(x,y)\frac{\partial}{\partial y}u(x,y) = s(x,y)$

- 2D-Diffusion: $\Delta u = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) u(x,y) = s(x,y)$

- (Viscous) fluid flow ($\dim + 1$ equations):

$$\begin{aligned} -\Delta u + \nabla p &= f \\ \operatorname{div} u &= 0 \end{aligned}$$

- Not PDEs:

$$u(x) = \int_\Omega u(y) g(x-y)\, dy$$

or

$$u(x) = r(x) u(x-c)$$

Partial Differential Equations – 3

- Important cases in practice show a plethora of parameters (domain, coefficients).

- The solution is a function, i.e. it lies in an infinite-dimensional space. The function space appropriate for a certain PDE problem can be very intricate.

- Existence and uniqueness of PDE solutions is often very difficult or even unknown.

Numerics for PDEs

These properties lead to difficulties in the numerical treatment of practically relevant problems:

- Complicated domains have to be described (CAD systems).
- High numbers of unknowns are necessary for approximating the continuous solution sufficiently well.
- The arising discrete equations may be ill-conditioned and difficult to solve.

The Finite Element Method

Idea:

- Write the PDE in variational form: find $u \in V$ with $a(u, v) = f(v)$ for all $v \in V$.
- Approximate the function space V with a space V_h made from piecewise polynomial functions defined on a mesh.
- Construct the discrete equations by restricting the variational form to V_h.

Properties: Flexibility, good theoretical foundation, OTOH somewhat more complex than e.g. finite-difference methods

Requirements for PDE software

- Formulating the problem in a convenient way
  $\rightsquigarrow$ domain-specific language

- Discretizing the problem $\rightsquigarrow$ controllable expert system

- Solving the discrete problem $\rightsquigarrow$ controllable expert system

- Postprocessing (?)

We want: Ease of use, flexibility, efficiency

Solving a PDE with Femlisp

The "fruit fly": Many important PDEs are variations and extensions of

$$
\begin{aligned}
-\Delta u(x) &= f(x) & \quad x \in \Omega \\
u(x) &= g(x) & \quad x \in \partial\Omega
\end{aligned}
$$

Easy choice of parameters: $\Omega = (0,1)^2$, $f \equiv 1$

Solving steps

- Create the PDE problem
- Write it to a blackboard, together with criteria what is needed
- Call solve with the blackboard
- Analyse the blackboard

The code

```
(let ((problem
       (create-problem '<cdr-problem>
           (:domain (n-cube-domain 2) :components '(u))
         (setup-coefficients (patch)
           (select-on-patch (patch)
             (:boundary
              (coeff FL.CDR::SCALAR-CONSTRAINT () 0.0))
             (:d-dimensional
              (list (coeff FL.CDR::DIFFUSION () (eye 2))
                    (coeff FL.CDR::SCALAR-SOURCE () 1.0))))))))
  (storing
    (solve (blackboard :problem problem
                       :success-if '(> :time 1.0)))))
```

Femlisp features

- n-dimensional meshes (simplex and simplex-product cells)

- Conforming finite elements of arbitrary order

- Duality-based error estimation and local mesh refinement

- Graphics (interface to OpenDX)

- Scalar equations, linear elasticity, Navier-Stokes

Why Common Lisp?

- Dynamic typing and REPL
- Good compilation to native code
- Macros, read macros, supporting embedded languages
- Stability (at least of the base language:-)
- History and reputation

My Road-to-Lisp

- 1991-1995: UG (=*unstructured grids*) in C, ugly license
- Around 1996: I considered using Emacs as an interface for C; I asked RMS about this and was directed towards Guile/Scheme
- Until 2000: I tried to create something similar to UG using a Guile/C combination.
- When starting to write documentation for it, I could not satisfactorily explain the Guile/C choice.

- Winter 2000: I asked in comp.lang.lisp if it was possible to make some simple array loops run reasonably fast and got promising answers.

- . . . since then I am happy with Common Lisp.

Femlisp compared to other PDE packages

- Interactive development, no edit-save-compile-run cycle
- No separation lines between user level and system level
- Small code base (Femlisp has only about 30.000 lines)
- Interesting related software: Axiom/Maxima, CL-MUPROC, Cells, Qi, ACL2, . . .

Performance comparisons?

- Difficult because of flexibility $\leftrightarrow$ performance
- How much flexibility is allowed?
- Who chooses the benchmark? DFG benchmark?
- How much time does a participant invest in a benchmark?
- How to evaluate?

Comparison with M++

Task: 3D linear elasticity problem: pull a cubic block in the vertical direction.

Results:

- Femlisp was faster than M++ for large problems (multigrid)
- Now: M++ is faster (and works on a cluster)
- But: I did invest almost no time
- And: we found two errors in M++ comparing convergence rates

Comparison with COMSOL Multiphysics (Femlab)

Task: Eigenvalues of a Maxwell problem in a dielectric medium

Results:

Femlab

- Easy to use (GUI), but calls to support necessary
- Only simplex meshes
- Reliability?

Femlisp

- Coding necessary (EVP)
- Rectangular meshes
- Not all problems implemented

About the same speed for comparable accuracy — due to structured meshes and high order discretisations.

Zooming into details of Femlisp

- Generally useful stuff: module `basic`
  - portability, AMOP, debugging, regression tests, multiprocessing
- Full Matrices: module `matlisp`
- FFI to SuperLU, UMFPACK, LAPACK
- Meshes
- Graphics (DX interface)

Somewhat tedious TODO items

- Remove bug in complex number LAPACK interface for Lispworks
- Sharper regression tests (speed and convergence rates)
- More interactive graphics, interface to VTK/Mayavi
- Interface to tetrahedral mesh generator (Tetgen)
- Making multigrid work together with adaptive refinement

Interesting TODO items

- Better (more functional? AI techniques?) solving strategy
- Parallelisation of iterative solvers on multi-threaded machines (up to now no real solving speedup)
- Real parallelisation on workstation clusters—Unobtrusively?
- Large-scale-low-cost calculations keeping vectors (maybe also the mesh) in a database—Unobtrusively? CL-MUPROC?
- High-performance calculations—introducing a "locally structured mesh"-FEM class

Wishlist

- Limit on the allowed memory with graceful exits
- OS threads for Allegro/Linux (Lispworks?)
- Fast floating-point arithmetic for OpenMCL
- SLIME with ECL, GCL
- More stable libraries (maybe)
- More stable packaging (Debian/Ubuntu)

Final remarks

- I never regretted switching to Common Lisp
- Recently I used CL also for web applications (thanks, Edi!)
  - Interactive math exercises:
    `http://www.matheum.de`
  - Web server for course administration:
    `http://www.vorlesungsverwaltung.de`