CHALMERS

FINITE ELEMENT CENTER

PREPRINT 2002-13

Object Oriented Implementation of a General Finite Element Code

Rickard Bergström

Chalmers Finite Element Center
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg Sweden 2002

CHALMERS FINITE ELEMENT CENTER

Preprint 2002-13

Object Oriented Implementation of a General Finite Element Code

Rickard Bergström

CHALMERS

Chalmers Finite Element Center
Chalmers University of Technology
SE–412 96 Göteborg Sweden
Göteborg, December 2002

Object Oriented Implementation of a General Finite Element Code
Rickard Bergström
NO 2002-13
ISSN 1404–4382

Chalmers Finite Element Center
Chalmers University of Technology
SE–412 96 Göteborg
Sweden
Telephone: +46 (0)31 772 1000
Fax: +46 (0)31 772 3595
`www.phi.chalmers.se`

Printed in Sweden
Chalmers University of Technology
Göteborg, Sweden 2002

Object Oriented Implementation of a General Finite Element Code

Rickard Bergström

Abstract

We present the framework of a finite element code, written with the explicit aim of being as general and as close to the mathematical viewpoint as possible. Due to the object oriented programming paradigm, the weak form of the mathematical model is easily implemented in a style close to the mathematical formula, and element types and quadrature rules can quickly be changed or easily implemented. This work only concerns the discretization of the problem, and does not consider mesh generation or solvers for the linear system.

1 Introduction

The finite element method (FEM) is a general framework for solving several classes of PDE:s and ODE:s, and there are several program packages that have implemented the method, both commercial and free. Many of these implementations are however written for a certain engineering application in, e.g., solid mechanics, and it may be difficult to adapt the code to a specific need. This paper describes a code which instead tries to stay close to the mathematical framework of FEM. The benefits include an accessible problem formulation and easy extension with new element types etc.

This implementation is made in C++, a language gaining popularity in the numerical society. Advances in compilers and programming techniques have made efficiency comparable to that of simpler structural languages such as Fortran. The possibility of creating new types, called objects, and specify their properties makes it possible to write programs which are highly readable and with the same notations as in the literature on the subject.

There are similar object oriented code projects with different generality and level of abstraction. In solid mechanics, large scale design is described in e.g. [11] and [21] and a smaller "idea testing" design is presented in [25]. A more general approach can be found in [13]. Also SIFFEA [15] is close to this projects but is written for 2D calculations. Another inspiration for this code has been the Norwegian commercial package DiffPack [1], also described in [18].

2 Aim of the code

When this coding project started, there was a need for a flexible finite element code for research purpose. Different research projects imposed the necessity for an easy change of the weak form in which the problem was posed, as well as the possibility of implementation of different types of elements. Multiphysics with coupled equations or hybrid methods was also a possible extension. Moreover, the different applications made it necessary to have a large range of algebraic solvers to use.

These specifications have to a large extent been met by the implementation presented in this paper. The weak form is isolated and easily readable. This makes the change of finite element formulation or the implementation of a new equation straightforward. Several different types of elements have been incorporated, including standard nodal elements of various polynomial order and the Nédélec edge elements.

Many different projects have started with the core of this code and later been transformed to more specific, and thus more fine tuned and efficient, codes. The applications include least-squares FEM for interface problems [8], discontinuous LSFEM [6][9], magnetostatic computations with coupled scalar and vector potentials [14], eddy current computations with edge elements [19][20] , hybrid FEM-FDM for the wave equation [5], and incompressible flow in porous media [23], as well as smaller in-house projects concerning, e.g., error estimators for higher order polynomial finite elements and eddy current/heat transfer coupled problems. A parallel version of the code has also been developed.

3 The abstraction of FEM

The strength of object oriented programming is that it is possible to implement abstractions of the underlying problem, which make the code easy to read, understand, and maintain. The code can be written in a way that makes the coupling to the underlying problem direct and apprehendable.

In the case of scientific computing, it is natural to use the objects that are present in the mathematical or physical formulation of the problem. On the highest level, it is the *equation* (or rather the weak form when it concerns finite element methods) together with *boundary conditions*, the *domain* and possibly algebraic or constitutive relations that further define the computational problem.

In the finite element method the *element* is the centerpoint. We will adapt to the definition of Brenner and Scott [12] for this, i.e. a finite element consists of

- a *subdomain* for the element,
- a function space defined on the domain, the *shape functions*,
- the *degrees of freedom*.

This is a general definition that includes all types of elements and methods.

Furthermore the concept of inner products, or *integration*, is needed to form the weak form.

3.1 Implementation issues in object oriented numerical programming

In scientific computing the efficiency of the code is an important issue. A high level of abstraction in the code may impede performance greatly. It is therefore necessary to compromise to get a code that is both general and readable as well as efficient. To keep inner loops free of time consuming function calls or other constructions, some of the abstract objects mentioned above have been collected into the same class to facilitate numerically efficient code. The different concepts are however clearly present.

4 Code details

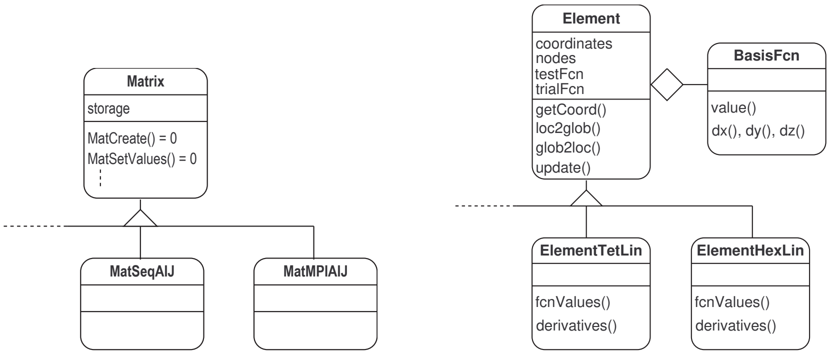
The components described in this section have been used directly in the main function of a C++ program. To hide details even further and make the usage cleaner, it would of course be possible to wrap the objects together in problem modules. One might thus see the program described in Section 5 as one part of a larger code.

The notation used in describing the code is based on that of Rumbaugh *et al.* [22]. To represent a class, we use a box divided in three sections. The first contains the class name, the second shows the class attributes, while the third lists the methods in the class. Object relationship is shown with lines connecting the class objects. A symbol on the line indicates the type and number of relationships: no symbol identify only one object, a solid circle termination means zero to many objects, diamonds indicate aggregation, i.e. the class is *made-up-of* the attached classes, and finally, triangles are used to indicate inheritance with the base of the triangle towards the derived class and the tip towards the parent class.

4.1 Extent of the code

In this paper, we are only concerned with finding the discrete finite element solution on a given geometry and mesh. Thus, no routines to generate meshes or post-processing (except for adaptive mesh refinement) are included. In practise, we have used either meshes from a separate program package (such as ProEngineer or Matlab) or written routines to generate meshes for a specific problem. Post-processing has been handled in the same way. Most of the classes to handle the meshes have been provided through another code project, related to DiffPack [1][18] and partly described in [7], while some have been developed for this project. For self-containment a brief description of the code used for grids, geometry and adaptivity is provided.

Furthermore, to have a large library of solvers to choose from, we have used the software package PETSc [4][24]. This package is written in object oriented C, and provides a nice interface to work with sparse matrices and vectors and the solution process of matrix problems. For the parallel version we used the METIS-package [2][16][17] to decompose the problem.



(a) *Matrix class.* (b) *Element class*

Figure 1: Description of the *Matrix* and *Element* classes. Vectors are handled analogously to *Matrix*.

4.2 General structure

4.2.1 Basic objects

Some basic mathematical objects are essential when numerically solving PDEs: matrices and vectors. We have chosen to use the implementation in PETSc. It provides a basic *Matrix* type which is the interface to different implementations, i.e. storage formats, of sparse matrices such as the traditional row oriented *MatSeqAIJ* or the blocked version *MatSeqBAIJ*. Parallel versions are also provided, e.g. *MatMPIAIJ* and *MatMPIBAIJ*. The *Vector* object is analogous.

4.2.2 Element

An *Element* object contains all three parts of the definition of a finite element cited above, following the discussion in Section 3.1. The parts are however separated in the class. The object includes one part containing the grid information concerning this particular element, i.e. the nodes and their coordinates. Note that there is not an *Element* object for each element in the grid, but the information in *Element* is continuously updated with geometrical information.

There is a *Basis function* object corresponding to the shape functions mentioned above. However, objects of this type are only used as an interface when communicating with the *Equation* class, the actual description of the function space is implemented in different instances of *Element*, such as quadratic polynomials on tetrahedrons, *ElementTetQuad*, or edge elements on hexahedrons, *ElementEdgeHex*. The shape functions are implemented for a reference element and operations on the functions are performed through the *Mapper*. Finally, a map for going from local to global degrees of freedom is needed.

4.2.3 Equation

In objects of this class, we include the weak form and routines to perform the integration needed to form these functionals. For some applications it would have been advantageous to isolate the integration, with the possible problem of time consuming function calls or overloaded operators in the inner loops. We have mainly considered integration by numerical quadrature and this construction has been flexible enough.

Using information from an element and a quadrature rule, the *Equation* has functionality to compute the element matrices by numerical integration. In order to do this, also element neighbor information is needed. Furthermore, the mapping from local to global degrees of freedom, provided by the element, is used to compute global matrix indices for the element matrix terms.

Different problems have been implemented in instances of *Equation*, e.g. *EquationLaplace* or *EquationHarmonicMaxwell*, in a form very similar to the analytical variational formulation, see the example in Section 5.

4.2.4 Boundary condition

Essentially, there are two methods implemented for boundary conditions. The first is a direct implementation of Dirichlet conditions, *BCStrong*. In this case we have chosen to keep the structure of the system matrix and not eliminate the unknowns corresponding to degrees of freedom on the Dirichlet boundary. In the case of a scalar Dirichlet condition, this consists of zeroing rows in the matrix and inserting ones in the diagonal and data values in the right hand side vector. When the condition involves normal or tangential traces, a local coordinate transformation has been implemented to be performed before operating on the matrix.

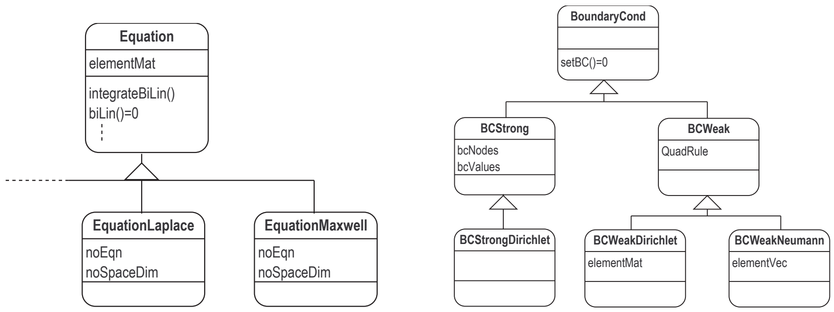
The second method, *BCWeak*, includes inhomogeneous natural boundary conditions and weakly imposed conditions. Then additional integrals are added to the weak form in a procedure identical to the one in *Equation*. For interface problems or the discontinuous Galerkin method, there are classes corresponding to *BCWeak* performing the same integration.

4.2.5 Quadrature rule

This class a straightforward implementation of the quadrature rules used. It isolates the implementation of the points for the quadrature rule and their corresponding weights.

4.2.6 Mapper

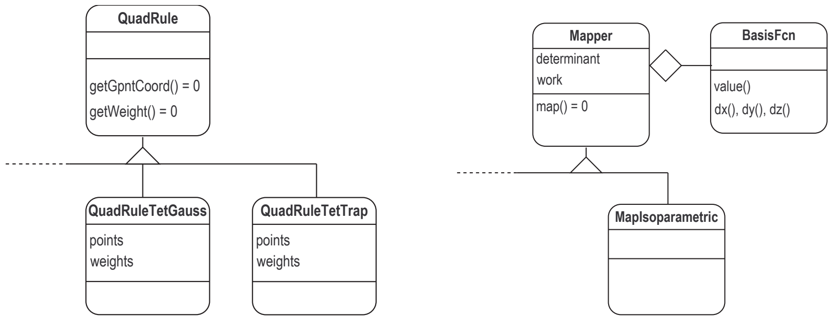
The *Mapper* takes care of the transformation from the actual element in the grid to a reference element. For a parametric mapping it uses an *Element* object to define the mapping and to update the values of the shape function in the quadrature points.



(a) *Equation class.*

(b) *BC class*

Figure 2: Description of the classes for the equation and boundary conditions.



(a) *QuadRule class.*

(b) *Mapper class*

Figure 3: Description of the *Quadrature rules* and *Mapper* classes.

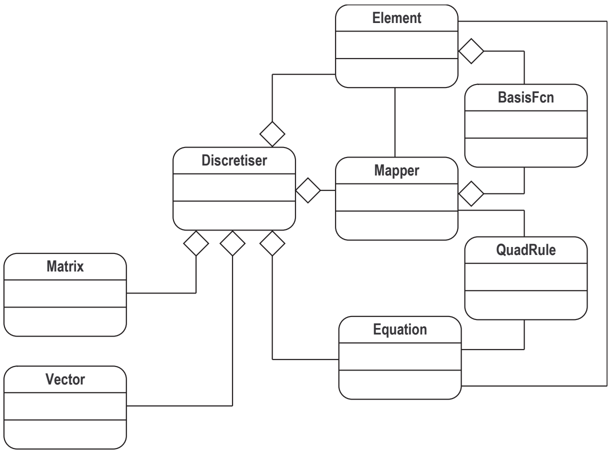


Figure 4: The relation between the *Discretiser* and other classes.

4.2.7 Discretiser

This is where the different parts are put together to assemble the system matrix. The *Element* is updated for each element in the grid, new function values are computed by the *Mapper*, element matrices are integrated in *Equation* and assembled into the *Matrix*. In the case of the parallel version of the code, this is also where partitioning of the problem is handled.

4.2.8 Solver

The *Solver* class we use is picked from the PETSc package. It contains a large range of Krylov subspace methods, together with different preconditioners. They are also implemented for parallel computations. We refer to the manual of PETSc [3] for more details.

Apart from the PETSc solvers, a multigrid solver has been implemented and tested, see [10].

4.2.9 Grid and geometry

A *Grid* object contains the elements, nodes, and coordinates of the nodes of the computational mesh. Elements and nodes are not implemented as separate classes but are represented as arrays in *Grid*. Derived from the base class is, e.g., objects to handle nested adaptively refined meshes, *GridFEHier*, and meshes for discontinuous/interface problems, *GridDiscont*. There is also a class for surface meshes to handle boundary conditions, *GridSurface*.

The geometry description is handled by NURB curves and surfaces, and collected into *GridGeometry*. The *GridGeometry* is used to compute correct normals on boundary and interface surfaces, and to make sure that a refined mesh respects the geometry by projecting new nodes onto the geometrical surface.

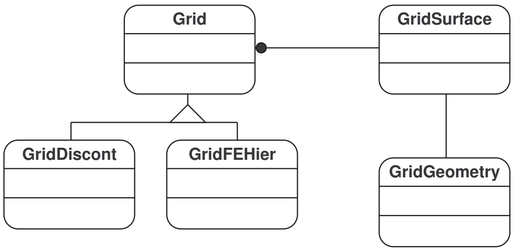


Figure 5: The relation between the classes that handle geometry and meshes.

5 An example: the Poisson problem

To get a better conception of how these classes interact in a code, we will present a computation of the Poisson problem, written in pseudo C++ language.

The starting point is to model the problem,

$$-\nabla \cdot A\nabla u = f \quad \text{in } \Omega, \tag{5.1a}$$

$$n \cdot A\nabla u = g_N \quad \text{on } \Gamma_N, \tag{5.1b}$$

$$u = g_D \quad \text{on } \Gamma_D. \tag{5.1c}$$

For simplicity we choose $f(x) = A(x) = 1$, $\Gamma_D = \partial\Omega$, and $g_D(x) = 0$. The variational formulation of the problem is then: find $u \in \mathcal{V}_0$ such that

$$(\nabla u, \nabla v) = (f, v), \quad \forall v \in \mathcal{V}_0, \tag{5.2}$$

or, writing out the functionals explicitly,

$$\int_\Omega \frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial v}{\partial y} + \frac{\partial u}{\partial z}\frac{\partial v}{\partial z}\, dx = \int_\Omega f v\, dx \tag{5.3}$$

This is implemented in the class *EquationPoisson* as

```
real f(){
  return 1.;
}

EquationPoisson:: biLin() {
  BasisFcn v = *testFcn
  BasisFcn u = *trialFcn

  biLin = u.x()*v.x()+u.y()*v.y()+u.z()*v.z()

}
```

```
EquationPoisson:: load() {
  BasisFcn v = *testFcn

  load = f()*v()

}
```

In the main function, the domain is loaded into the *Grid* and *GridGeometry* classes. Then equation, element, and quadrature rule are chosen before the *Discretiser* is called to create the linear system of equations. Finally the *Solver* generates the approximate solution by solving the matrix problem. This will look like

```
main(){

  // Set up the problem
  int quadOrder = 4;
  int noSpaceDim = 3;
  Grid                grid;
  GridGeometry        geom;
  QuadRuleTetGauss    quad(quadOrder);
  ElementTetLin       element;
  EquationPoisson     equation(noSpaceDim);
  BCStrongDirichlet   bc;
  MapIsoparametric    mapper(noSpaceDim);
  Discretiser         discrete;
  SolverGMRES         solver(discrete);
  Vector              U;

  // Read mesh and geometry
  geom.scan('geom.file');
  grid.getSurface().attachGeom( geom );
  grid.scan('grid.file');

  // Create and assemble the system matrix and rhs vector
  discrete.createSeqProblem( grid, equation );
  discrete.discretise( equation, bc, mapper, element, quad, grid );

  // Solve the matrix problem
  U=solver.solve();

}
```

As we can see, the *Discretiser* is the kernel of this program. The member function `discretise` performs the actual assembling of the problem and looks like

```
Discretiser:: discretise( Equation eq,
                          BC bc,
                          Mapper map,
                          Element elm,
                          QuadRule quad,
                          Grid grid){

  for( el=0; el < grid.getNoElms(); el++ ){
    elm.update( el, grid );
    map.map( quad, elm );
    eq.integrateBiLin( quad, elm, elmMat );
    MatSetValues( elmMat, A );
    eq.integrateLoad( quad, elm, elmVec );
    VecSetValues( elVec, b );
  }
  bc.setBC( A, b )

}
```


Acknowledgements

We would like to thank Dr. Klas Samuelsson for providing the grid classes and the multigrid solver, and Dr. Mårten Levenstam for interesting discussions and helpful suggestions.

This work is supported by ABB Corporate Research, Sweden, and the Swedish Foundation for Strategic Research.

References

[1] Diffpack home page. `http://www.nobjects.com/Products/Diffpack`.

[2] Metis home page. `http://www-users.cs.umn.edu/~karypis/metis/`.

[3] S. Balay, W.D. Gropp, L.C. McInnes, and B.F. Smith. PETSc 2.0 users manual. Technical Report ANL-95/11 - Revision 2.0.28, Argonne National Laboratory, 2000.

[4] S. Balay, W.D. Gropp, L.C. McInnes, and B.F. Smith. PETSc home page. `http://www.mcs.anl.gov/petsc`, 2000.

[5] L. Beilina, K. Samuelsson, and K. Åhlander. A hybrid method for the wave equation. Preprint 14, Chalmers Finite Element Center, Chalmers University of Technology, 2001.

[6] R. Bergström. Least-squares finite element method with applications in electromagnetics. Preprint 10, Chalmers Finite Element Center, Chalmers University of Technology, 2002.

[7] R. Bergström, A. Bondeson, C. Johnson, M.G. Larson, Y. Liu, and K. Samuelsson. Adaptive finite element methods in electromagnetics. Technical Report 2, Swedish Institute of Applied Mathematics (ITM), 1999.

[8] R. Bergström and M.G. Larson. Discontinuous least-squares finite element methods for the Div-Curl problem. Preprint 12, Chalmers Finite Element Center, Chalmers University of Technology, 2002.

[9] R. Bergström and M.G. Larson. Discontinuous/continuous least-squares finite element methods for elliptic problems. Preprint 11, Chalmers Finite Element Center, Chalmers University of Technology, 2002. To appear in Math. Models Methods Appl. Sci.

[10] R. Bergström, M.G. Larson, and K. Samuelsson. The $\mathcal{L}\mathcal{L}^*$ finite element method and multigrid for the magnetostatic problem. Preprint 2, Chalmers Finite Element Center, Chalmers University of Technology, 2001.

[11] J. Besson and R. Foerch. Large scale object-oriented finite element code design. *Comput. Methods Appl. Mech. Engrg.*, 142:165–187, 1997.

[12] S.C. Brenner and L.R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer-Verlag, 1994.

[13] P. Donescu and T.A. Laursen. A generalized object-oriented approach to solving ordinary and partial differential equations using finite elements. *Finite Elements in Analysis and Design*, 22:93–107, 1996.

[14] M. Fontana. Adaptive finite element computation of 3D magnetostatic problems in potential formulation. Preprint 8, Chalmers Finite Element Center, Chalmers University of Technology, 2001.

[15] X. Jiao, X.Y. Li, and X. Ma. SIFFEA: Scalable integrated framework for finite element analysis. In S. Matsuoka, R.R. Oldehoeft, and M. Tholburn, editors, *Computing in Object-Oriented Parallel Environments*, pages 84–95. Springer-Verlag, 1999.

[16] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[17] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. *SIAM Rev.*, 41(2):278–300, 1999.

[18] H.P. Langtangen. *Computational Partial Differential Equations*. Springer-Verlag, 1999.

[19] Y.Q. Liu, A. Bondeson, R. Bergström, M.G. Larson, and K. Samuelsson. Edge element computations of eddy current in laminated materials. Submitted to IEEE Trans. Magn., 2002.

[20] Y.Q. Liu, A. Bondeson, K. Samuelsson, R. Bergström, M.G. Larson, and C. Johnson. Eddy current computations using adaptive grids and edge elements. *IEEE Trans. Mag.*, 38:449–452, 2002.

[21] B. Patzák and Z. Bittnar. Design of object oriented finite element code. *Advances in Engineering Software*, 32:759–767, 2001.

[22] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[23] N.H. Sharif and N.-E. Wiberg. Adaptive ICT-procedure for non-linear seepage flows with free surface in porous media. *Communications in Numerical Methods in Engineering*, 18:161–176, 2002.

[24] G. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition*. Cambridge University Press, 1996.

[25] T. Zimmerman, P. Bomme, D. Eyheramendy, L. Vernier, and S. Commend. Aspects of an object-oriented finite element environment. *Computers and Structures*, 68:1–16, 11998.

Chalmers Finite Element Center Preprints

2001–01 *A simple nonconforming bilinear element for the elasticity problem*
Peter Hansbo and Mats G. Larson

2001–02 *The $\mathcal{LL}^*$ finite element method and multigrid for the magnetostatic problem*
Rickard Bergström, Mats G. Larson, and Klas Samuelsson

2001–03 *The Fokker-Planck operator as an asymptotic limit in anisotropic media*
Mohammad Asadzadeh

2001–04 *A posteriori error estimation of functionals in elliptic problems: experiments*
Mats G. Larson and A. Jonas Niklasson

2001–05 *A note on energy conservation for Hamiltonian systems using continuous time finite elements*
Peter Hansbo

2001–06 *Stationary level set method for modelling sharp interfaces in groundwater flow*
Nahidh Sharif and Nils-Erik Wiberg

2001–07 *Integration methods for the calculation of the magnetostatic field due to coils*
Marzia Fontana

2001–08 *Adaptive finite element computation of 3D magnetostatic problems in potential formulation*
Marzia Fontana

2001–09 *Multi-adaptive galerkin methods for ODEs I: theory & algorithms*
Anders Logg

2001–10 *Multi-adaptive galerkin methods for ODEs II: applications*
Anders Logg

2001–11 *Energy norm a posteriori error estimation for discontinuous Galerkin methods*
Roland Becker, Peter Hansbo, and Mats G. Larson

2001–12 *Analysis of a family of discontinuous Galerkin methods for elliptic problems: the one dimensional case*
Mats G. Larson and A. Jonas Niklasson

2001–13 *Analysis of a nonsymmetric discontinuous Galerkin method for elliptic problems: stability and energy error estimates*
Mats G. Larson and A. Jonas Niklasson

2001–14 *A hybrid method for the wave equation*
Larisa Beilina, Klas Samuelsson and Krister Åhlander

2001–15 *A finite element method for domain decomposition with non-matching grids*
Roland Becker, Peter Hansbo and Rolf Stenberg

2001–16 *Application of stable FEM-FDTD hybrid to scattering problems*
Thomas Rylander and Anders Bondeson

2001–17 *Eddy current computations using adaptive grids and edge elements*
Y. Q. Liu, A. Bondeson, R. Bergström, C. Johnson, M. G. Larson, and K. Samuelsson

2001–18 *Adaptive finite element methods for incompressible fluid flow*
Johan Hoffman and Claes Johnson

2001–19 *Dynamic subgrid modeling for time dependent convection–diffusion–reaction equations with fractal solutions*
Johan Hoffman

2001–20 *Topics in adaptive computational methods for differential equations*
Claes Johnson, Johan Hoffman and Anders Logg

2001–21 *An unfitted finite element method for elliptic interface problems*
Anita Hansbo and Peter Hansbo

2001–22 *A P^2–continuous, P^1–discontinuous finite element method for the Mindlin-Reissner plate model*
Peter Hansbo and Mats G. Larson

2002–01 *Approximation of time derivatives for parabolic equations in Banach space: constant time steps*
Yubin Yan

2002–02 *Approximation of time derivatives for parabolic equations in Banach space: variable time steps*
Yubin Yan

2002–03 *Stability of explicit-implicit hybrid time-stepping schemes for Maxwell's equations*
Thomas Rylander and Anders Bondeson

2002–04 *A computational study of transition to turbulence in shear flow*
Johan Hoffman and Claes Johnson

2002–05 *Adaptive hybrid FEM/FDM methods for inverse scattering problems*
Larisa Beilina

2002–06 *DOLFIN - Dynamic Object oriented Library for FINite element computation*
Johan Hoffman and Anders Logg

2002–07 *Explicit time-stepping for stiff ODEs*
Kenneth Eriksson, Claes Johnson and Anders Logg

2002–08 *Adaptive finite element methods for turbulent flow*
Johan Hoffman

2002–09 *Adaptive multiscale computational modeling of complex incompressible fluid flow*
Johan Hoffman and Claes Johnson

2002–10 *Least-squares finite element method with applications in electromagnetics*
Rickard Bergström

2002–11 *Discontinuous/continuous least-squares finite element methods for Elliptic Problems*
Rickard Bergström and Mats G. Larson

2002–12 *Discontinuous least-squares finite element methods for the Div-Curl problem*
Rickard Bergström and Mats G. Larson

2002–13 *Object oriented implementation of a general finite element code*
Rickard Bergström

These preprints can be obtained from

www.phi.chalmers.se/preprints