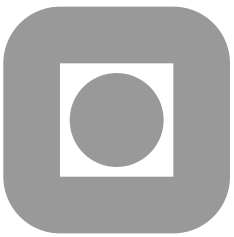
NORGES TEKNISK-NATURVITENSKAPELIGE
UNIVERSITET

The Design and Implementation of a Spectral Element Code

by

Bjarte Hægland and Bård Skaflestad

PREPRINT
NUMERICS NO. 3/2004



NORWEGIAN UNIVERSITY OF SCIENCE AND
TECHNOLOGY
TRONDHEIM, NORWAY

This report has URL `http://www.math.ntnu.no/preprint/numerics/2004/N3-2004.ps`
Address: Department of Mathematical Sciences, Norwegian University of Science and Technology,
N-7491 Trondheim, Norway.

Abstract

We describe the design and implementation of a spectral element code. We assume familiarity with basic finite element methods and do not explain all terms encountered in the text. Furthermore, a thorough introduction to the theory of these methods has been related to the references. The emphasis of this report is on preliminary results gained from the actual implementation and design decisions made so far.

The code is still a work in progress and additional features and capabilities will be forthcoming.

1 Introduction

Discretisation of partial differential equations by high-order polynomial basis functions was consistently developed and studied by Patera and coworkers at Massachusets Institute of Technology in the second half of the 1980's [1, 2]. Some of the main features of these methods are tractability of geometrically realistic physical domains, locally structured and globally unstructured meshes, a focus on tensor product forms, operator evaluation without explictly forming the discrete operators and favorable convergence properties—especially for problems with smooth solutions.

The methods are based on a Galerkin type weak formulation of the differential equation. Discrete problems are constructed by choosing finite dimensional subspaces of suitable test and trial function spaces, and evaluating the various linear and bilinear forms using high-order numerical quadrature. The finite dimensional subspaces are defined as follows

- Find $N+1$ Gauss–Lobatto–Legendre (GLL) nodes $\{\xi_i\}_{i=0}^N$, defined as the zeros of

$$(1-x^2)L'_N(x), \quad x \in [-1,1]$$

with L_N denoting an N-th degree Legendre polynomial, not necessarily normalised.

- Construct $N+1$ Lagrangian interpolating polynomials $\ell_j(x) : [-1,1] \to \mathbb{R}$. Expressing $\ell_j(x)$ in terms of $L_N(x)$ and using fundamental properties of the latter, $\ell_j(x)$ is given by

$$\ell_j(x) = -C_j \frac{(1-x^2)L'_N(x)}{x-\xi_j}, \quad C_j = \frac{1}{N(N+1)} \cdot \frac{1}{L_N(\xi_j)}.$$

We note that the numbers $D_{ij} = \ell'_j(\xi_i)$ play a fundamental rôle in the approximation of derivatives.

- Construct a reference element $\hat{\Omega} = \otimes_{j=1}^d [-1,1] = [-1,1]^d$. Define the polynomial space $I\!P_N(\hat{\Omega})$ consisting of polynomials defined on $[-1,1]^d$ having maximum polynomial degree N in each space direction. A basis for $I\!P_N(\hat{\Omega})$ is given by

$$\big\{\otimes_{j=1}^d \{\ell_0, \ldots, \ell_N\}\big\}.$$

- Divide the computational domain $\Omega \subset \mathbb{R}^d$ into K non-overlapping subdomains or elements, $\Omega_k, k = 1, \ldots, K$, and for each subdomain define basis functions $\varphi_{\mathbf{i}}^k = \hat{\varphi}_{\mathbf{i}} \circ \mathcal{F}_k^{-1}$. The function $\mathcal{F}_k$ maps the reference element $\hat{\Omega}$ uniquely onto Ω_k.

- Let $I\!P_N(\Omega_k) = \{v : v \circ \mathcal{F}_k \in I\!P_N(\hat{\Omega})\}$ denote the space of polynomials of maximum degree N in each space direction defined on Ω_k. Then the discrete space X_N is defined by

$$X_N = I\!P_N(\Omega) = \{v \in X : v\big|_{\Omega_k} \in I\!P_N(\Omega_k),\ k = 1, \ldots, K\}$$

in which X is one of the test or trial function spaces alluded to above.

Using these tools, an abstractly formulated discrete problem can be stated as: Find $u_N \in X_n$ such that

$$a_N(u_N, v_N) = \mathcal{F}_N(v_N) \tag{1}$$

for all $v_N \in Y_N$. Here, Y_N is a discrete space of test functions constructed similarly to X_N. The subscript N on the bilinear form $a_N(\cdot,\cdot)$ and linear form $\mathcal{F}_N(\cdot)$ signifies that all continuous integrals are replaced by numerical quadrature. The Gaussian quadrature rules are derived from GLL nodes $\{\xi_i\}_{i=0}^N$ and will, in exact arithmetic, integrate all elements of $\mathbb{P}_{2N-1}(\Omega)$ exactly.

We note that the discrete subspaces X_N and Y_N embody all imposed essential boundary conditions, whereas natural boundary conditions are absorbed into the linear form $\mathcal{F}_N(\cdot)$. This practice is entirely equivalent to what is found in low-order finite element methods. Furthermore, using Krylov subspace methods for the resulting linear systems, we do not need to explicitly form the discrete operators. The action of an operator on a vector is effectuated through procedure calls and, taking advantage of a local data representation, usually entails a sequence of matrix-matrix multiplications followed by direct stiffness summation and masking of boundary nodes to enforce continuity and essential boundary conditions.

A thorough overview of the theory of high-order element methods, their application to fluid flow problems, and software development based on this technology can be found in [3].

2 Implementation and software development

The current code base consists of approximately 10,000 lines of `Fortran 95` code, divided into nearly 40 source files. `Fortran 95` is well suited to implementing compute intensive numerical algorithms. While upholding the `FORTRAN 77` tradition of producing efficient machine code, `Fortran 95` added, among other features, dynamic memory allocation, array expressions reminiscent of `MATLAB`, derived types and generic subroutines. Together with the concept of *modules*, these tools enable rapid development, maintainability and extensibility to the code base.

In addition to the `Fortran 95` computational core is an input file parser, written in the `C` programming language, for the coarse mesh input file. The input file is generated by a third-party geometry preprocessor in the format used by the commercial product NEKTON$^{\text{TM}}$.

The code is still in active development so little tuning and statement level optimisation has been implemented. Furthermore, additional features not available at present will be needed in the immediate future.

2.1 Overview of the code

As currently implemented, the code can be divided into five basic subsystems or functionality classes. In summary these subsystems are

- Basic tools such as special purpose matrix-matrix multiply subroutines, determining GLL nodes and quadrature weights in the associated quadrature rule $Q_N[f] = \sum_{j=0}^N \rho_j f(\xi_j)$, generation of one-dimensional differentiation and interpolation operators, and nodal numbering.

  Most of this functionality is provided by a library of `FORTRAN 77` subroutines implemented by Professor E.M. Rønquist for another spectral element code [4].

- Input and spectral element mesh generation.

- Derived datatypes, memory management, global data objects, additional routines for initialising and finalising these objects.

- Discrete differential operators, interpolation and restriction operators, preconditioning operators.

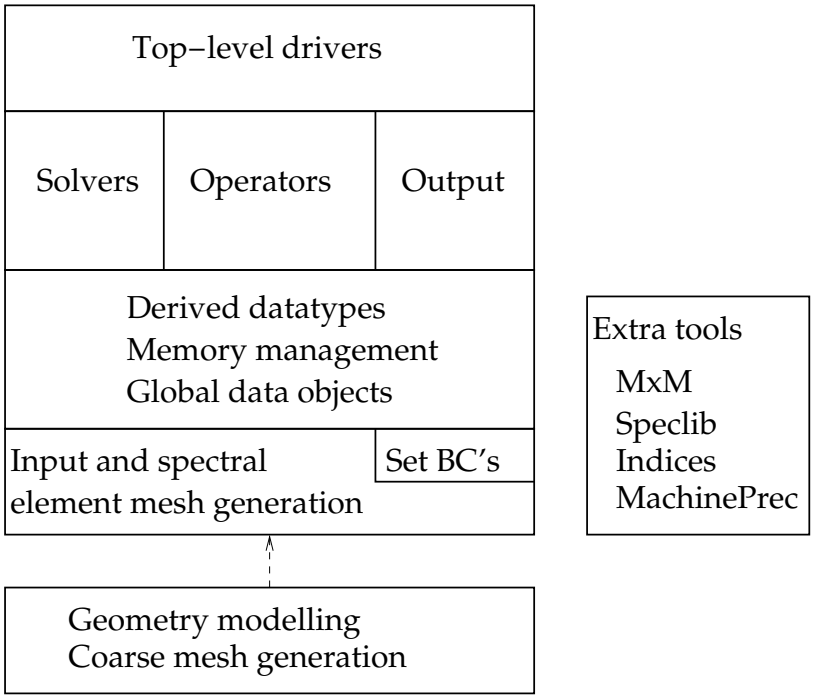


Figure 1: Overview of the spectral element code

- Solvers for linear systems, specialised solvers for the Poisson and Stokes problems.

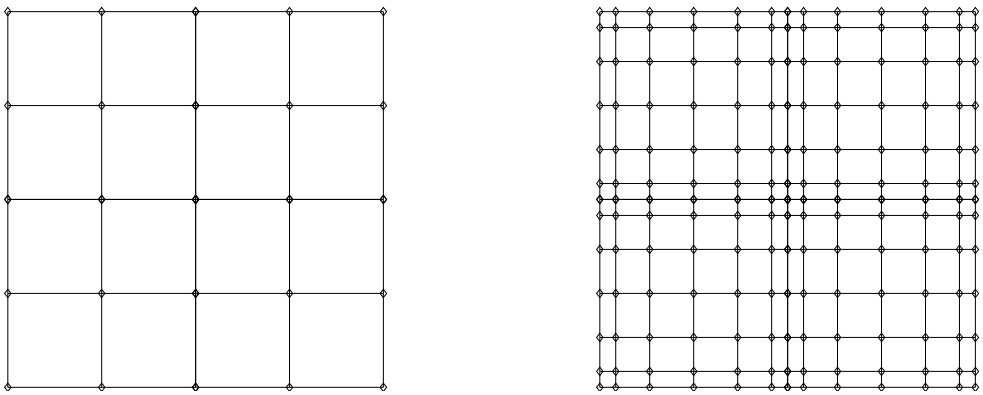
Finally, a top-level driver takes care of starting the subsystems in correct order, outputting results and shutting down the subsystems upon termination. Figure 1 depicts the idealised relationships amongst the subsystems. Some layering violations do, however, occur in the actual code. We note in particular that setting boundary conditions is done in conjunction with reading the input mesh file as boundary values are specified in this file.

2.2 Generating the spectral element mesh

Discretising the computational domain with spectral elements is in principle equivalent to the problem solved in low-order finite element methods. There are many finite element mesh generators available, both commercial and free, with varying capabilities and sophistication. However, there are few mesh generators capable of producing a spectral element mesh.

A method due to Gordon and Hall [5] can be used in defining the spectral elements, but using this method effectively requires detailed knowledge about the domain boundary. This knowledge, commonly known as the geometric model of the physical domain, may not be readily available in the output from most mesh generators.

To work around this problem we use a third-party finite element mesh generator capable of producing second order quadrilateral based finite elements. Assuming all element edges are represented by parabolas, the element nodes can be defined by polynomial interpolation of edges and faces, and, in three space dimensions, in volumes. Figure 2 shows the result obtained when interpolating a second order two-dimensional mesh with four elements up to polynomial degree six. Within each spectral element we number the nodes locally with so-called *natural numbering*. This numbering is essential in spectral element codes, as it enables efficient operator evaluation based on tensor products.



(a) Finite element mesh

(b) Spectral element mesh

Figure 2: Constructing a high-order element mesh from initial low-order element descriptions.

2.3 Field variables

A scalar field $u(\mathbf{x}) = \sum_{i=1}^{\mathcal{N}} u_i \varphi_i(\mathbf{x})$, where $\mathcal{N}$ is the total number of degrees of freedom, one usually represents u by its nodal values $u_i,\ i = 1, \ldots, \mathcal{N}$, stored in a vector. There are at least two different strategies of storing these values. A straightforward approach is to represent u as the vector $\mathbf{u} = \left[u_1, u_2, \ldots, u_{\mathcal{N}}\right]^T$. This technique is usually employed in finite element codes.

Alternatively one may represent u using a so-called *local representation*. In a local data representation, the nodal values for one element is stored as one contiguous region in a large vector $\mathbf{u}_L = \left[u_1^1, u_2^1, \ldots, u_{\mathcal{N}_1}^1, u_1^2, u_2^2, \ldots, u_{\mathcal{N}_2}^2, \ldots, u_1^K, u_2^K, \ldots, u_{\mathcal{N}_K}^K\right]^T$. K is the number of elements and $\mathcal{N}_k$ is the number of nodes within element k. This way of representing u allows for easy access to the data for one particular element, but on the other hand is slightly inefficient with respect to memory consumption. Function values corresponding to element nodes on internal element boundaries are stored redundantly in all elements sharing the nodes. This effect is, however, ameliorated by the fact that the ratio of surface nodes to volume nodes is approximately $1/N$ in $I\!P_N$ elements, irrespective of spatial dimension d.

The global data representation uses the least amount of memory, but as the polynomial degree increases, the redundancy of memory of the local representation becomes less severe compared to the total memory consumption. We will be using the local data representation for the scalar field u. Similarly a vector field is represented as a single long vector. The first part of the vector is the local representation of the first component of the vector field, akin to a scalar field variable. The second part is the local representation of the second component and so on.

Both scalar fields and vector fields are represented internally in the code as general *field variables*. A field variable is a vector with some additional information. In addition to a pointer to the data, a field variable also knows the the number of components of the field and the polynomial degree of the mesh it is defined over.

In many programming tasks we need to access the same part of memory from within different parts of the code. To accomplish this, we have created a global lists of field variables. The list will grow or shrink as needed, using dynamic memory allocation. A field variable may be accessed either by name or an identification number. Field variables are the main data objects in the code, and consume the most memory resources.

2.4 Operators

In every finite element code one needs to evaluate operators of the weak form. These could be differential operators such as the Laplacian (∇^2) or a convective operator of the form ($\mathbf{U} \cdot \nabla$).

Ordinary finite element codes usually assemble these operators and store the resulting numbers in a suitable datastructure. Banded or sparse matrix storage are prevalent options. This is driven by relatively low matrix bandwidth for low-order finite element. As the polynomial degree increases, so does the bandwidth and the matrices will soon become too large and reduce the computational efficiency and numerical accuarcy of the code.

For this reason, no operators, apart from coarse grid preconditioning operators, are assembled into their matrix form. Operator evaluation is instead effectuated by subroutines computing the *action* of the operator. Other operators could be related to interpolation and differentiation among many. All operators accept a field variable as input and return a field variable.

3 The Poisson problem

In this section we will concentrate on the Poisson problem

$$\begin{aligned} -\nabla^2 u &= f(\mathbf{x}), \quad \mathbf{x} \in \Omega \\ u &= g(\mathbf{x}), \text{ on } \partial\Omega_E \\ \frac{\partial u}{\partial n} &= h(\mathbf{x}), \text{ on } \partial\Omega_N, \end{aligned} \tag{2}$$

where $\partial\Omega_E \cap \partial\Omega_N = \emptyset$.

3.1 Variational form

The weak formulation of the Poisson problem reads: Find $u \in X = H_0^1(\Omega)$ such that

$$a(u, v) = \mathcal{F}(v), \quad \forall v \in X,$$

where $H_0^1(\Omega) = \{v \in H^1(\Omega) : \ v\big|_{\partial\Omega_N} = 0\}$ and

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\Omega, \quad \mathcal{F}(v) = \int_\Omega f v \, \mathrm{d}\Omega + \int_{\partial\Omega_N} g v \, \mathrm{d}\Gamma.$$

The discrete problem reads: Find $u_N \in X_N = \mathbb{P}_N \cap X$ such that

$$a_N(u, v) = \mathcal{F}_N(v), \quad \forall v \in X_N.$$

The subscript N signifies that continuous integrals are replaced by quadrature rules.

3.2 Preconditioning

It is well known that the matrix representation of the Laplacian operator $a(\cdot, \cdot)$ is poorly conditioned in a spectral element discretisation. Availability of good preconditioning schemes is therefore essential when solving the resulting linear system using an iterative method.

We choose a simplified iterative substructuring method suggested by Rønquist in [6]. This particular method avoids the computation of a Schur complement solution, or, more precisely, uses an inexact solver for the interface values. The method decomposes the the discrete space X_N as:

$$X_N = X_{N_0} + \sum_{k=1}^{K} X_k + X_\Gamma.$$

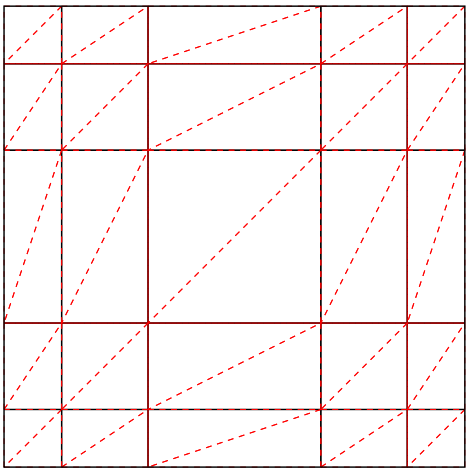


Figure 3: Triangulation of a spectral element

Here, X_{N_0} is the space associated with a coarse discretisation of the Poisson problem. N_0, the polynomial degree determining the resolution of the coarse grid discretisation, is constrained to the interval $[1, N)$ and is usually fixed at $N_0 = 2$. The subspaces X_k are associated with individual spectral elements Ω_k and are defined as

$$X_k = \{v \in I\!P_N(\Omega_k) : v\big|_{\partial\Omega_k} = 0\}$$

for all elements $\Omega_k = 1, \ldots, K$. The space X_Γ is defined as

$$X_\Gamma = \{v \in X_N : v = v\big|_\Gamma\},$$

where Γ is the collection of all internal element interfaces and any external boundaries with prescribed natural boundary conditions.

For each of the subspaces we follow the approach of [6] and explicitly assemble a discrete Laplacian operator. The coarse grid Laplacian operator is specified by the matrix A_0. Similarly, A_k is the Laplacian operator associated with the local problem over each element Ω_k. This matrix is constructed as a low-order finite element discretisation within the single spectral element. Figure 3 shows the imposed finite element mesh on a single spectral element of polynomial degree 5 in two space dimensions. The resulting discrete Laplacian is referred to as $A_{k,\mathrm{FE}}$. Finally, $A_\Gamma = \operatorname{diag} A\big|_\Gamma$ is the diagonal elements of the high-order Laplacian matrix A associated with nodes on Γ.

The matrices for the coarse problem and the local element problems are all banded, symmetric, and positive definite. Inverting such matrices can be done using special purpose Cholesky factorisation subroutines from the LAPACK library. The interface matrix A_Γ is diagonal and thus easily inverted. The interface problem solved is a simple approximation to the Schur complement on the subdomain interfaces as explained in [7].

The resulting preconditioner to the operator A is

$$B^{-1} = R_0 A_0^{-1} R_0^T + \sum_{k=1}^{K} R_k A_k^{-1} R_k^T + R_\Gamma A_\Gamma^{-1} R_\Gamma^T$$

where R_0, R_k, and R_Γ are prolongation operators for the coarse problem, the local problems and the interface problem respectively. The restriction operators R_0^T, $R_k^T, k = 1, \ldots, K$, and R_Γ^T ensure that the subproblems are defined on the corresponding subdomains only.

N	N_{it}	$\|r\|_{L^2}$	N	N_{it}	$\|r\|_{L^2}$
2	1	$0.000 \cdot 10^{-00}$	17	15	$0.818 \cdot 10^{-15}$
3	1	$0.263 \cdot 10^{-15}$	18	16	$0.558 \cdot 10^{-15}$
4	3	$0.606 \cdot 10^{-17}$	19	15	$0.693 \cdot 10^{-15}$
5	3	$0.229 \cdot 10^{-16}$	20	16	$0.663 \cdot 10^{-15}$
6	6	$0.203 \cdot 10^{-18}$	21	16	$0.228 \cdot 10^{-15}$
7	6	$0.351 \cdot 10^{-17}$	22	16	$0.719 \cdot 10^{-15}$
8	10	$0.514 \cdot 10^{-21}$	23	16	$0.320 \cdot 10^{-15}$
9	10	$0.476 \cdot 10^{-18}$	24	16	$0.764 \cdot 10^{-15}$
10	13	$0.516 \cdot 10^{-15}$	25	16	$0.346 \cdot 10^{-15}$
11	13	$0.539 \cdot 10^{-16}$	26	16	$0.859 \cdot 10^{-15}$
12	16	$0.740 \cdot 10^{-16}$	27	16	$0.335 \cdot 10^{-15}$
13	14	$0.792 \cdot 10^{-15}$	28	16	$0.840 \cdot 10^{-15}$
14	16	$0.261 \cdot 10^{-15}$	29	16	$0.331 \cdot 10^{-15}$
15	15	$0.347 \cdot 10^{-15}$	30	16	$0.739 \cdot 10^{-15}$
16	16	$0.458 \cdot 10^{-15}$			

Table 1: Number of conjugate gradient iterations used in solving the Poisson problem for a single spectral element. N is polynomial degree, N_{it} is iteration count in the PCG algorithm, and $\|r\|_{L^2}$ is the experimentally obtained residual when specifying an absolute residual error tolerance of $\|r\|_{L^2} < \text{Atol} = 10^{-15}$.

3.3 Numerical examples for the Poisson problem

In this section we investigate the effect of preconditioning on the convergence rate of our Poisson solver.

3.3.1 Validation of preconditioner

In this section we will choose $f \equiv 1$ and $g \equiv 0$ and $\partial\Omega_E = \partial\Omega$ in (2).

To validate the local preconditioning we choose $\Omega = [0,1]^2$, $f \equiv 1$, $g \equiv 0$ and $\partial\Omega_E = \partial\Omega$ in (2). Table 1 shows the number of iterations used in the conjugated-gradient algorithm used on the preconditioned system. Clearly the number of iterations is bounded from above and independent of polynomial degree N.

To validate the coarse grid preconditioner we choose $N = N_0$. In this case $A_{N_0} = A$ and the conjugate-gradient algorithm should converge in only one iteration. Table 2 shows the number of CG iterations for a varying number of elements and confirms the expected efficiency of the coarse grid preconditioner. As a final test of the preconditioning we solve the Poisson problem of this section using $K = 16$ elements. The number of CG iterations with and without preconditioning is included in Table 3. Preconditioning clearly impacts the convergence rate of the conjugate gradient algorithm. For the preconditioned case the number of iterations depend linearly on the polynomial degree N. This is as expected.

3.3.2 Convergence results

Choosing the forcing term f, and the essential boundary conditions g in (2), such that the exact solution is

$$u(x,y) = x(1 - \mathrm{e}^{10(x-1)})y(1 - \mathrm{e}^{10(y-1)}), \tag{3}$$

we study the convergence with respect to the polynomial degree N on a fixed number of spectral elements.

K	N_{it}	$\|r\|_{L^2}$
1	1	$0.000 \cdot 10^{-00}$
4	1	$0.145 \cdot 10^{-15}$
9	1	$0.247 \cdot 10^{-15}$
16	1	$0.175 \cdot 10^{-15}$
25	1	$0.352 \cdot 10^{-15}$
36	1	$0.453 \cdot 10^{-15}$
49	1	$0.532 \cdot 10^{-15}$
64	1	$0.545 \cdot 10^{-15}$
81	1	$0.672 \cdot 10^{-15}$
100	1	$0.839 \cdot 10^{-15}$

Table 2: Coarse grid preconditioning when $N = N_0$. K is number of elements in the mesh, N_{it} is number of iterations used in the conjugate-gradient algorithm and the third column is obtained residual.

N	$N_{\text{it,CG}}$	$N_{\text{it,PCG}}$	N	$N_{\text{it,CG}}$	$N_{\text{it,PCG}}$
2	10	11	17	313	125
3	22	26	18	341	133
4	39	36	19	370	138
5	58	42	20	399	144
6	75	49	21	429	152
7	93	57	22	458	158
8	110	64	23	488	162
9	130	70	24	519	168
10	150	77	25	552	172
11	171	84	26	586	178
12	192	91	27	620	183
13	215	98	28	655	189
14	239	105	29	690	193
15	263	112	30	727	201
16	287	118			

Table 3: Number of conjugate gradient iterations with ($N_{\text{it,PCG}}$) and without ($N_{\text{it,CG}}$) preconditioning. N is polynomial degree. $K = 16$ elements was used in all tests and error residual tolerance set to $\|r\|_{L^2} < \text{Atol} < 10^{-5}$.

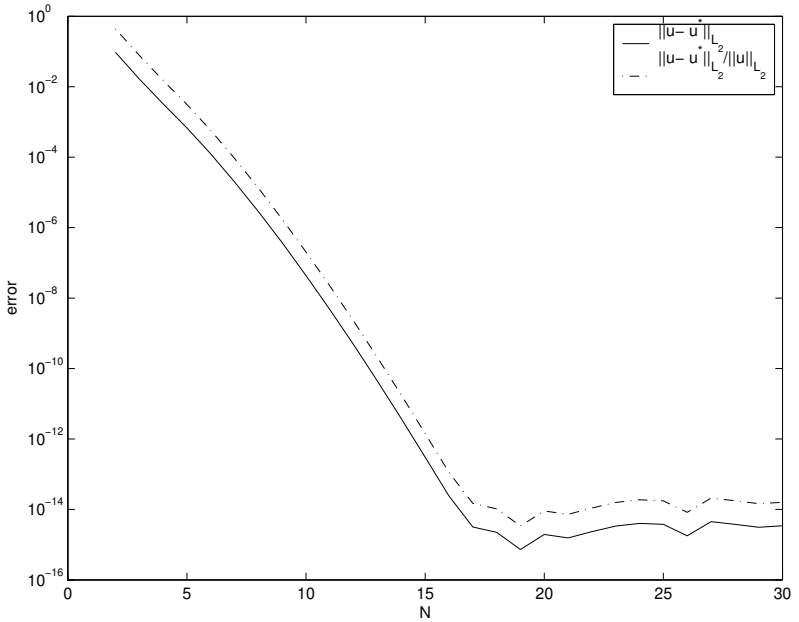


Figure 4: Convergence of Poisson problem

As pointed out in [6], the function (3) represents a boundary layer solution to the Poisson equation as $x, y \to 1$. The function cannot be represented exactly by tensor product polynomial basis functions, and is thus a good test case for the convergence of the numerical method.

Table 4 shows the absolute error and relative error in the solution to this problem as a function of the polynomial degree N. This test case was perfomed on $K = 4$ elements. Figure 4 depicts the same convergence history data visually. We note that both the absolute and relative errors decrease exponentially fast as the polynomial degree increases.

4 Steady Stokes

In a domain $\Omega \subset \mathbb{R}^d$, the steady Stokes problem of incompressible fluid flow is given by the following set of differential equations.

$$-\Delta \mathbf{u} + \nabla p = \mathbf{f}, \tag{4}$$

$$\operatorname{div} \mathbf{u} = 0, \tag{5}$$

Here, $\mathbf{u}$ is the velocity and p is the pressure. We assume prescribed velocity boundary conditions, $\mathbf{u} = \mathbf{g}$, on the entire boundary $\partial\Omega$.

Following [3], equations (4)–(5) are rewritten in a weak formulation as: Find $\mathbf{u}$ in $H_0^1(\Omega)^d$ and p in $L_0^2(\Omega)$ such that

$$a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, p) = (\mathbf{f}, \mathbf{v}), \quad \forall\, \mathbf{v} \in H_0^1(\Omega)^d, \tag{6}$$

$$b(\mathbf{u}, q) = 0, \qquad \forall\, q \in L_0^2(\Omega), \tag{7}$$

where the bilinear forms $a(\cdot, \cdot)$ and $b(\cdot, \cdot)$ are given by

$$a(\mathbf{v}, \mathbf{w}) = (\nabla \mathbf{v}, \nabla \mathbf{w}),$$
$$b(\mathbf{v}, q) = -(\operatorname{div} \mathbf{v}, q).$$

The inner product $(\cdot, \cdot)$ used above is the standard L^2 inner product.

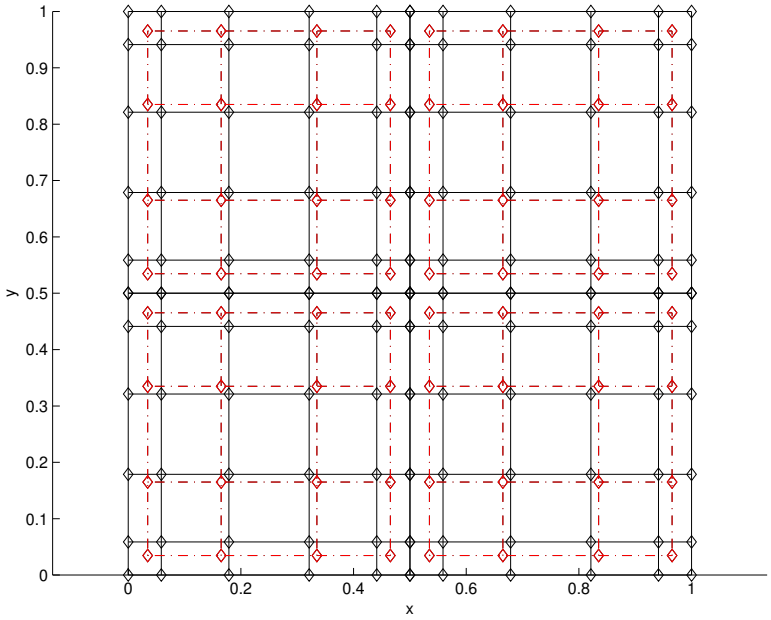


Figure 5: Staggered grid for $N = 5$

4.1 Discrete problem

We propose to use finite dimensional function spaces based on the $\mathbb{P}_N$–$\mathbb{P}_{N-2}$ spectral elements (see figure 5), with the pressure p interpolated at the internal Gauss–Legendre points and the velocity $\mathbf{u}$ interpolated at the standard Gauss–Lobatto–Legendre points to ensure a continuous velocity field. As noted in [3], the resulting pressure field is not necessarily continuous using this element. Using discrete spaces

$$X_N = \{\mathbf{v} \in H_0^1(\Omega)^d : \mathbf{v}|_{\Omega_k} \in \mathbb{P}_N^d(\Omega_k),\ k = 1, \ldots, K\}$$
$$Z_N = \{q \in L_0^2(\Omega) : q|_{\Omega_k} \in \mathbb{P}_{N-2}(\Omega_k),\ k = 1, \ldots, K\},$$

the discrete problem is given by: Find $(\mathbf{u}_N, p_N) \in (X_N, Z_N)$ such that

$$a_N(\mathbf{u}_N, \mathbf{v}_N) + b_N(\mathbf{v}_N, p_N) = (\mathbf{f}, \mathbf{v})_N, \quad \forall \mathbf{v}_N \in X_N, \tag{8}$$
$$b_N(\mathbf{u}_N, q_N) = 0, \qquad \forall q_N \in Z_N. \tag{9}$$

The discrete inner product and bilinear forms are given by

$$\begin{aligned} a_N(\mathbf{v}, \mathbf{w}) &= (\nabla \mathbf{v}, \nabla \mathbf{w})_{N,\mathrm{GLL}}, \\ b_N(\mathbf{v}, q) &= -(\operatorname{div} \mathbf{v}, q)_{N,\mathrm{GL}}, \end{aligned} \tag{10}$$

with subscripts GLL and GL representing evaluation at Gauss–Lobatto–Legendre and Gauss–Legendre nodes respectively. We note that this discrete problem is positive definite with respect to the velocity variables and negative definite with respect to the pressure variable. As such, the problem is an instance of a *saddle point problem* for which many algorithms have been proposed.

The pressure variable p_N is not really the same as the thermodynamic pressure, but rather corresponds to Lagrangian multipliers used in enforcing the incompressibility constraint (9) of the velocity. Without any restriction, p_N is known up to an additive constant only. However, choosing $p_N \in Z_N$ guarantees a unique pressure as the mean pressure level is zero.

The *Uzawa* pressure decoupling is a method for solving the resulting linear systems. In this method, we first compute p_N by solving the linear system

$$S\mathbf{p} = \mathbf{b} \tag{11}$$

where the Uzawa operator S and right hand side $\mathbf{b}$ are given by

$$S = \sum_{i=1}^{d} D_i A^{-1} D_i^T, \quad \text{and} \quad \mathbf{b} = -\sum_{i=1}^{d} D_i [A^{-1}(M\mathbf{f}_i - \tilde{A}\mathbf{u}_b^i) + \mathbf{u}_b^i],$$

respectively. Here, A is the discrete scalar Laplacian operator associated with the internal degrees of freedom. In other words, nodal values corresponding to nodes associated with essential boundaries conditions are excluded. The matrix $\tilde{A}$ is the discrete scalar Laplacian operator associated with all nodal point in the domain including nodes associated with essential boundary conditions. Furthermore, the diagonal matrix M is the mass matrix associated with the GLL velocity nodes.

The local derivative operators $D_i,\ i = 1, \ldots, d$ correspond to the action of the divergence operator on component i of a velocity field. Additionally, D_i^T is component i of the pressure gradient operator. These operators incorporate the effect of the geometry mappings $\mathcal{F}_k$.

The vector $\mathbf{u}_b^i$ is a vector containing Dirichlet boundary values of velocity component i for all $i = 1, \ldots, d$. We remark that all internal degrees of freedom have been masked out in this vector. The term $\tilde{A}\mathbf{u}_b^i$ is then the restriction to the internal nodes of the vector resulting from an application of the global, unrestricted Laplacian operator. In other words, it is the effect of the inhomogeneous essential boundary conditions acting on the internal degrees of freedom in the domain.

The Uzawa operator S is symmtric and positive definite. Thus, the linear system (11) can be solved using the preconditioned conjugate gradient algorithm. Furthermore, S is spectrally close to $\tilde{M}$, the diagonal mass matrix associated with the GL quadrature nodes and weights. Hence $\tilde{M}^{-1}$ is a suitable preconditioner for the conjugate gradient process. The application of S to a field variable w can be effected as shown in equations (5.3.13)–(5.3.15) of [3].

We note that each application of A^{-1} corresponds to solving a linear system of the form

$$A\mathbf{z} = \mathbf{y}$$

which is the same system as the one arising from discretising the scalar Poisson equation. We will solve this using homogeneous Dirichlet type boundary conditions on all boundaries. We remark that we use the same preconditioning for the Laplacian problem as used for the Poisson equation.

Having computed the pressure from (11), the velocity $\mathbf{u}$ is determined by solving d additional elliptic problems given by

$$A\mathbf{u}_i = D_i^T \mathbf{p} + M\mathbf{f}_i - \tilde{A}\mathbf{u}_b^i, \quad i = 1, \ldots, d$$

4.2 Numerical examples for the Stokes problem

In this section we will look at some test cases for the Stokes solver. These cases all have known analytic solutions. As an initial validation, we verified that the code will correctly reproduce the trivial solution $\mathbf{u} = \mathbf{0}, p = 0$ to the problem given by (4)–(5) with $\mathbf{f} = \mathbf{0}$ and $\mathbf{g} = \mathbf{0}$.

We remark that the test case with inhomogenous boundary conditions is artificial as we need to impose inhomogenous Dirichlet type boundary conditions on the whole external boundary to obtain a well-defined linear system. Specifying physically realistic boundary conditions in the general case requires reformulating the basic fluid flow problem (4)–(5) in the more general *stress formulation* of fluid flow. At present, the code does not support this feature.

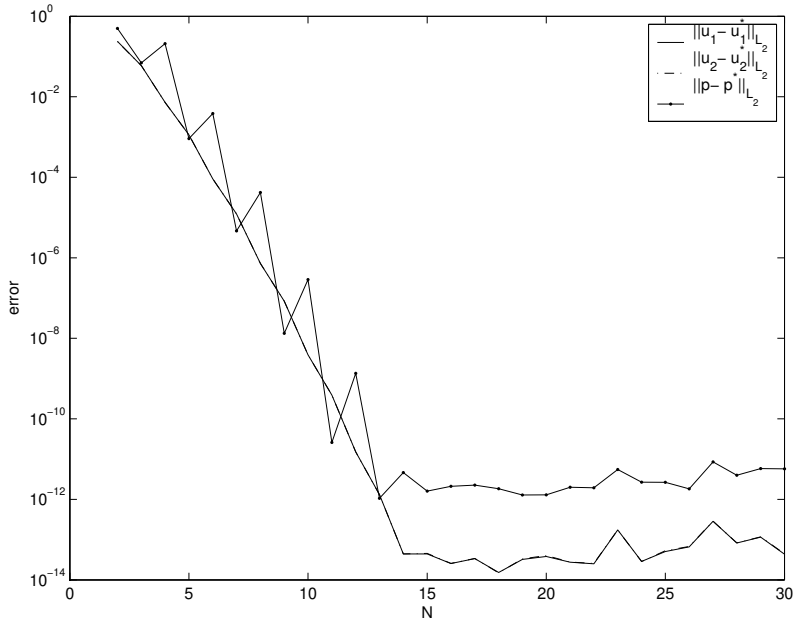


Figure 6: Convergence results for homogeneous test case for Stokes.

4.2.1 A homogeneous test problem

Let $\Omega = [0,1]^2$ and choose the forcing term $\mathbf{f}$ such that the exact solution of (4)–(5) is

$$\mathbf{u}(x,y) = \begin{bmatrix} \pi \sin(2\pi x)\sin^2(\pi y) \\ -\pi \sin(2\pi y)\sin^2(\pi x) \end{bmatrix}$$
$$p(x,y) = \cos(\pi x)\sin(\pi y),$$

corresponding to homogeneous Dirichlet boundary conditions $\mathbf{g} = \mathbf{0}$. We solve the system (10) on a grid with $K = 4$ elements and report the error in figure 6. The solution converges exponetially fast as the polynomial degree N increases. Figures 7 and 8 show, respectively, the velocity and pressure solutions obtained for polynomial degree $N = 9$ in this case. We remark that the pressure is plotted over the Gauss–Legendre grid, which means that no continuity can be enforced across element boundaries and that pressure values are undefined on element boundary nodes.

4.2.2 An inhomogeneous test problem

The example in this section is taken from [8], chapter 4. Let $\Omega = [0,1]^2$ and choose the forcing term $\mathbf{f}$ and the essential boundary conditions $\mathbf{g}$ such that the exact solution of (4)–(5) is

$$\mathbf{u}(x,y) = \begin{bmatrix} -\sin(xy)\, x \\ \sin(xy)\, y \end{bmatrix}$$
$$p(x,y) = \cos(xy) - a$$

where $a = \frac{1}{|\Omega|}\int_\Omega \cos(xy)\,\mathrm{d}\Omega$ and $|\Omega| = \int_\Omega 1\,\mathrm{d}\Omega$. We solve the system (10) on a grid with $K = 4$ elements and report the error in figure 9. The solution converges exponentially fast as the polynomial degree increases. Figures 10 and 11 show, respectively, the velocity and pressure solutions obtained for polynomial degree $N = 9$ in this case. We remark that the pressure is plotted over the Gauss–Legendre grid.

N	$\|u-u_N\|_{L^2}$	$\frac{\|u-u_N\|_{L^2}}{\|u\|_{L^2}}$	N	$\|u-u_N\|_{L^2}$	$\frac{\|u-u_N\|_{L^2}}{\|u\|_{L^2}}$
2	$0.943 \cdot 10^{-01}$	$0.440 \cdot 10^{+00}$	17	$0.317 \cdot 10^{-14}$	$0.148 \cdot 10^{-13}$
3	$0.166 \cdot 10^{-01}$	$0.773 \cdot 10^{-01}$	18	$0.223 \cdot 10^{-14}$	$0.104 \cdot 10^{-13}$
4	$0.326 \cdot 10^{-02}$	$0.152 \cdot 10^{-01}$	19	$0.728 \cdot 10^{-15}$	$0.339 \cdot 10^{-14}$
5	$0.672 \cdot 10^{-03}$	$0.314 \cdot 10^{-02}$	20	$0.194 \cdot 10^{-14}$	$0.906 \cdot 10^{-14}$
6	$0.123 \cdot 10^{-03}$	$0.573 \cdot 10^{-03}$	21	$0.156 \cdot 10^{-14}$	$0.725 \cdot 10^{-14}$
7	$0.198 \cdot 10^{-04}$	$0.924 \cdot 10^{-04}$	22	$0.234 \cdot 10^{-14}$	$0.109 \cdot 10^{-13}$
8	$0.285 \cdot 10^{-05}$	$0.133 \cdot 10^{-04}$	23	$0.337 \cdot 10^{-14}$	$0.157 \cdot 10^{-13}$
9	$0.370 \cdot 10^{-06}$	$0.172 \cdot 10^{-05}$	24	$0.401 \cdot 10^{-14}$	$0.187 \cdot 10^{-13}$
10	$0.436 \cdot 10^{-07}$	$0.203 \cdot 10^{-06}$	25	$0.376 \cdot 10^{-14}$	$0.175 \cdot 10^{-13}$
11	$0.472 \cdot 10^{-08}$	$0.220 \cdot 10^{-07}$	26	$0.178 \cdot 10^{-14}$	$0.831 \cdot 10^{-14}$
12	$0.471 \cdot 10^{-09}$	$0.219 \cdot 10^{-08}$	27	$0.450 \cdot 10^{-14}$	$0.210 \cdot 10^{-13}$
13	$0.436 \cdot 10^{-10}$	$0.203 \cdot 10^{-09}$	28	$0.379 \cdot 10^{-14}$	$0.177 \cdot 10^{-13}$
14	$0.376 \cdot 10^{-11}$	$0.176 \cdot 10^{-10}$	29	$0.312 \cdot 10^{-14}$	$0.145 \cdot 10^{-13}$
15	$0.304 \cdot 10^{-12}$	$0.142 \cdot 10^{-11}$	30	$0.342 \cdot 10^{-14}$	$0.159 \cdot 10^{-13}$
16	$0.242 \cdot 10^{-13}$	$0.113 \cdot 10^{-12}$			

Table 4: Convergence results for Poisson problem. N is the polynomial degree, $\|u-u_N\|_{L^2}$ is the absolute error in L^2 norm, and $\|u-u_N\|_{L^2}/\|u\|_{L^2}$ is the relative error in L^2 norm.

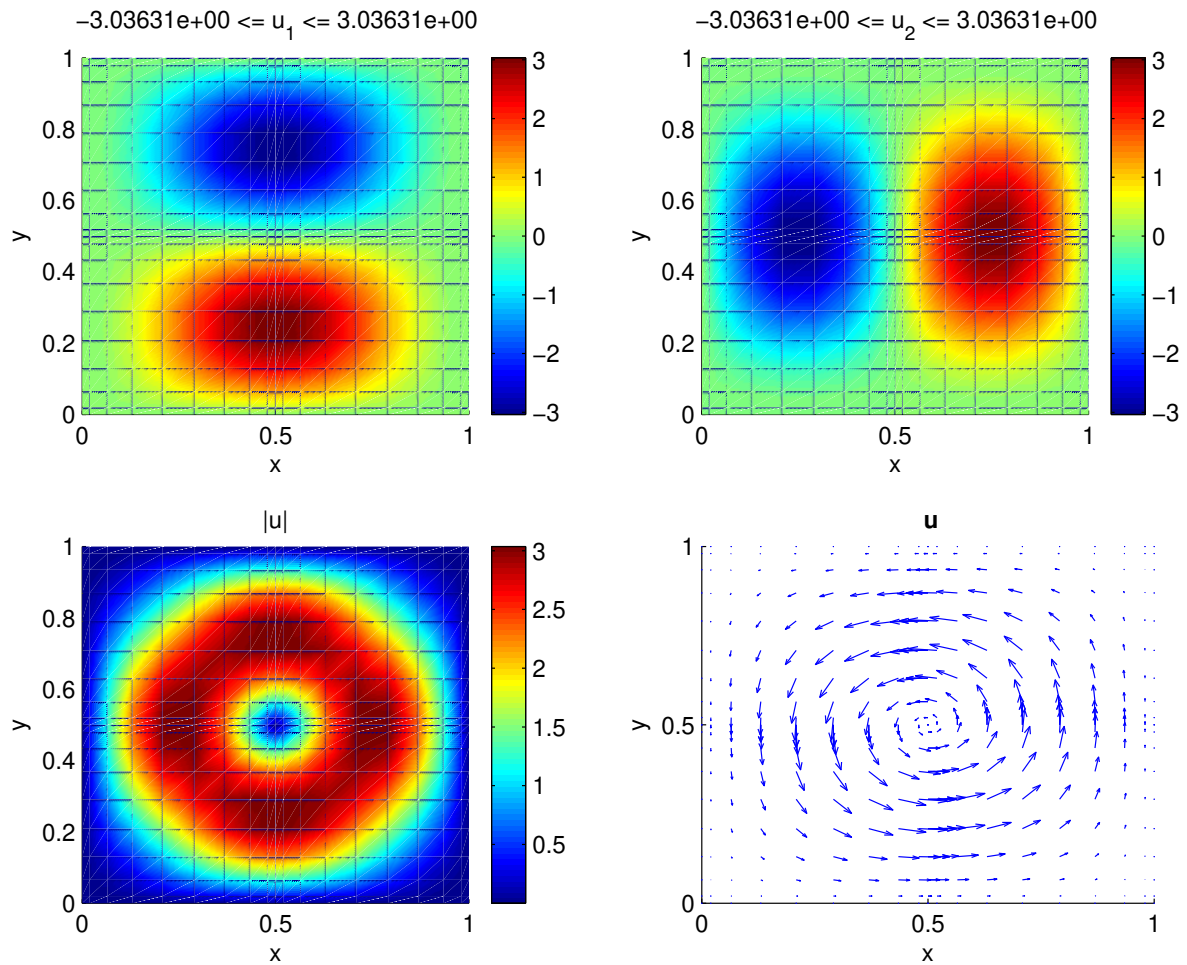


Figure 7: Velocity field for homogeneous Stokes test problem.

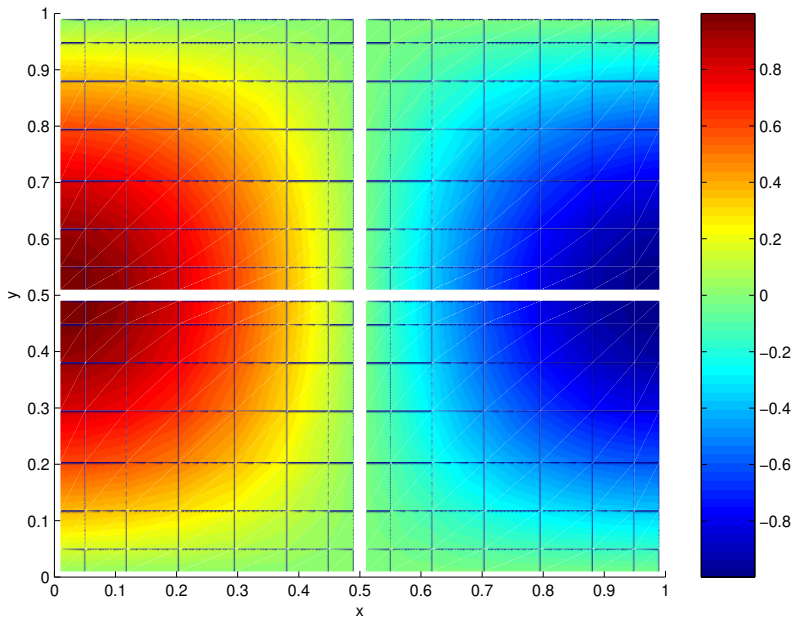


Figure 8: Pressure field for homogeneous Stokes test problem.

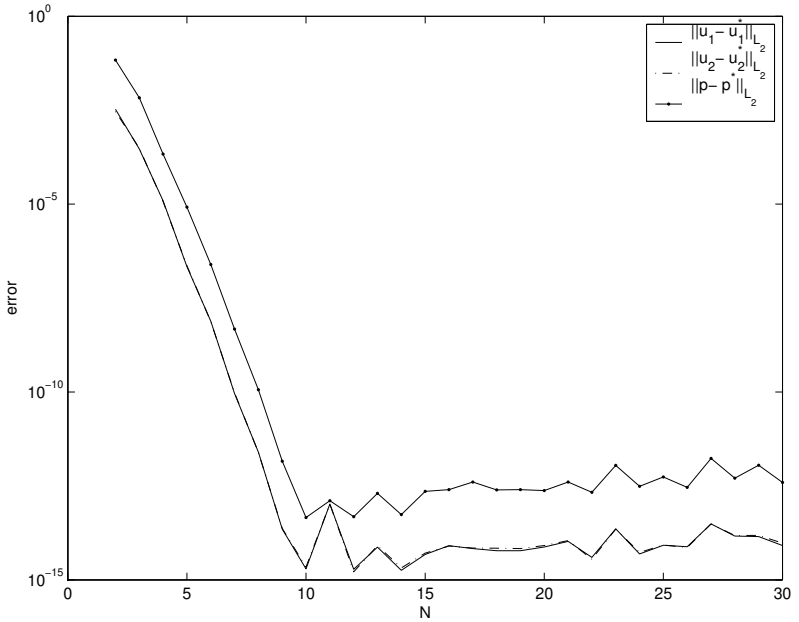


Figure 9: Convergence results for inhomogeneous Stokes model problem.

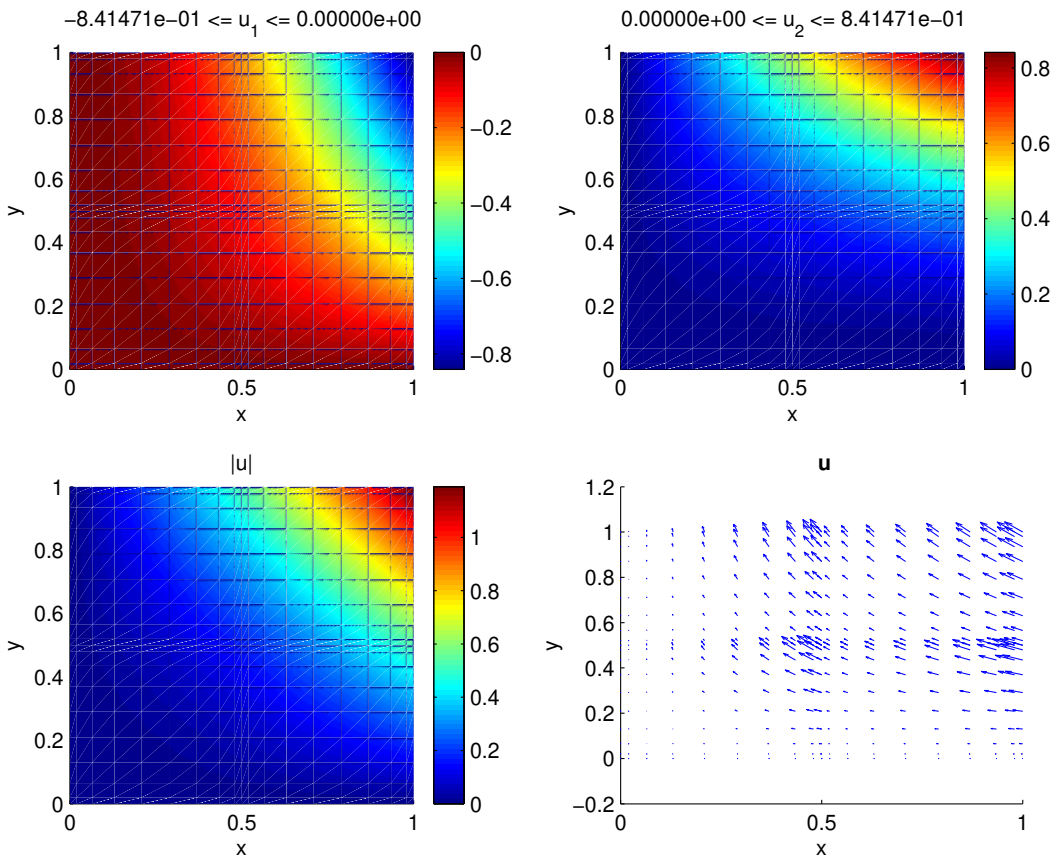


Figure 10: Velocity field for inhomogeneous Stokes model problem.

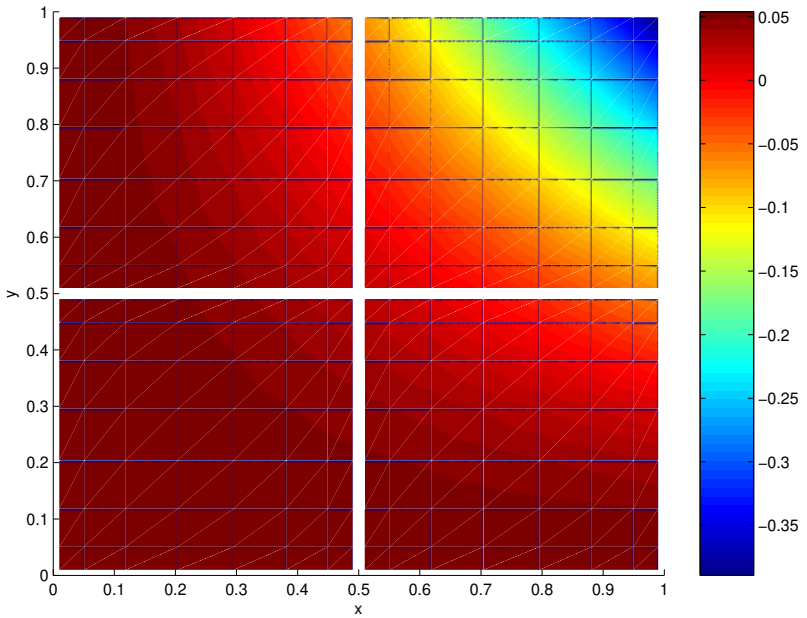


Figure 11: Pressure field for inhomogeneous Stokes model problem.

5 Future work

As stated previously, the code is a work in progress and is currently under heavy development. The following is a short list of features planned and research projects enabled by the addition of these features.

At present, the code is only able to solve Poisson and steady Stokes problems. However, much of the necessary groundwork has been put in and solving time dependent incompressible fluid flow problems governed by the Navier–Stokes equations is the focus of the coming work. Extending this to allow the general stress formulated model of incompressible fluid flow will cater to more realistic and general boundary conditions.

Adding the ability to solve mild heat conduction problems requires the Boussinesq variable density approximation and, possibly, tensor diffusivities.

As problem sizes increases there is also a need to add parallelism in order to get access to massively parallell processing systems. Communication needs are mostly limited to enforcing continuous solutions across element boundaries which means that adding some degree of parallelism should be reasonably straightforward.

Finally, we intend to study the effects of employing various advanced time stepping schemes in conjunction with high-order spatial discretisation.

References

[1] C. Canuto, Y. Hussaini, A. Quarteroni, and T.A. Zang. *Spectral Methods in Fluid Dynamics*. Springer Series in Computational Physics. Springer Verlag, Berlin, 1988.

[2] E.M. Rønquist. *Optimal Spectral Element Methods for the Unsteady Three-Dimensional Incompressible Navier–Stokes Equations*. PhD thesis, Massachusets Institute of Technology, Cambridge, MA, 1988.

[3] M.O. Deville, P.F. Fischer, and E.H. Mund. *High-Order Methods for Incompressible Fluid Flow*, volume 9 of *Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, 2003.

[4] E.M. Rønquist. The Design of a Finite Element/Spectral Element Code for Incompressible Fluid Flow. In M.E. Henderson, C.R. Anderson, and S.L. Lyons, editors, *Proceedings of the 1998 Workshop on Object Oriented Methods for Interoperable Scientific and Engineering Computing*, IBM TJ Watson Research Center, 1999. SIAM.

[5] W.J. Gordon and C.A. Hall. Construction of Curvilinear Co-ordinate Systems and Application to Mesh Generation. *International Journal for Numerical Methods in Engineering*, 7:461–477, 1973.

[6] E.M. Rønquist. A Domain Decomposition Solver for the Steady Navier–Stokes Equations. In A. Ilin and R. Scott, editors, *Proceedings of the Third International Conference on Spectral and High-Order Methods*. Huston J. of Mathematics, 1996.

[7] B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[8] H.P. Langtangen and A. Tveito, editors. *Advanced Topics in Computational Partial Differential Equations*, volume 33 of *Lecture Notes in Computational Science and Engineering*. Springer Verlag, 2003.