INFORMATICA, 2019, Vol. 30, No. 2, 243–268

DOI: http://dx.doi.org/10.15388/Informatica.2019.205

Efficient MATLAB Codes for the 2D/3D Stokes Equation with the Mini-Element

Jonas KOKO
LIMOS, Université Blaise Pascal – CNRS UMR 6158,
ISIMA, Campus des Cézeaux – BP 10125, 63173 Aubière cedex, France
e-mail: jonas.koko@uca.fr

Received: February 2018; accepted: December 2018

Abstract. We propose a fast MATLAB implementation of the mini-element (i.e. $P1$-Bubble/$P1$) for the finite element approximation of the generalized Stokes equation in 2D and 3D. We use cell arrays to derive vectorized assembling functions. We also propose a Uzawa conjugate gradient method as an iterative solver for the global Stokes system. Numerical experiments show that our implementation has an (almost) optimal time-scaling. For 3D problems, the proposed Uzawa conjugate gradient algorithm outperforms MATLAB built-in linear solvers.

Key words: finite element method, Stokes problem, Uzawa conjugate gradient, MATLAB.

1. Introduction

MATLAB is a popular problem solving environment, widely used for general scientific computation in education, engineering and research. MATLAB is nowadays a standard tools in many areas. Thanks to its collection of direct (e.g. LU, $LDL^\top$, Cholesky) and iterative (e.g. conjugate gradient, GMRES, bi-conjugate gradient) solvers, it is tempting to use MATLAB for the numerical approximation of partial differential equations (PDEs). For the finite element method (FEM), the main obstacle for using MATLAB is the assembly of the global matrix and vector. Since MATLAB built-in solvers are optimized, the assembly operations can take up to 99% of whole CPU time, as shown in Koko (2007), using an implementation with standard loop over triangles (Alberty *et al.*, 1999, 2002; Kwon and Bang, 2000) directly derived from compiled languages (FORTRAN and C/C++). Unfortunately, some vectorized FEM codes are less flexible and require a huge amount of memory due to the allocation of auxiliary arrays and the corresponding matrix operations (Koko, 2007; Rahman and Valdman, 2013, 2015). Recently, Koko (2016) proposed a MATLAB implementation close to the standard form by using cell-arrays to store the gradient of the basis functions, for the Poisson equation and linear elasticity in 2D and 3D.

In this paper, we propose a fast MATLAB implementation of the $P1$-Bubble/$P1$ finite element (*Mini element*, Arnold *et al.*, 1984; Ern and Guermond, 2002; Glowinski, 2003) for the generalized Stokes problem in 2D and 3D. The mini element for spatial discretization of the Stokes problem is easy to use in engineering practice since it

allows for the use of equal-order interpolation (the same mesh for velocity and pressure). Equal-order interpolation is very useful in large-scale multi-physics codes. Indeed, a code dealing with several independent variables (e.g. chemical species, velocity components, etc.) requires the transfer of information between its different components at each time-step. Fast implementation means that our code operates on array and does not use `for`-loops over elements (triangles of tetrahedrons) for the assembling operations. Instead, we use cell-arrays to store element matrices as in Koko (2016). We also propose a solution strategy for the final linear system. Indeed, we propose an efficient (preconditioned) Uzawa conjugate gradient method derived from the one used with $P2/P1$ (or $P1$-iso-$P2/P1$) finite element pair (Cahouet and Chabard, 1988; Glowinski, 2003). The proposed conjugate gradient method operates on the dual (pressure) space and, at each iteration, d independent linear systems are solved ($d = 2, 3$). Our implementation needs only MATLAB basic distribution functions and can be easily modified and refined.

The paper is organized as follows. The model problem is described in Section 2, followed by a finite element discretization in Section 3. The element matrices in 2D/3D are described in Section 4. In Section 5, we propose our Uzawa conjugate gradient method for solving the Stokes system. MATLAB implementation details are given in Section 6. Numerical experiments are carried out in Section 7. Readers can download and edit the codes from `http://www.isima.fr/~jkoko/Codes/KSTOK.tar.gz`.

2. The Model Problem

Let Ω be a bounded domain in $\mathbb{R}^d$ $(d = 2, 3)$ with a Lipschitz-continuous boundary Γ. Consider in Ω the Stokes problem

$$\alpha \boldsymbol{u} - \nu \Delta \boldsymbol{u} + \nabla p = \boldsymbol{f}, \quad \text{in } \Omega, \tag{2.1}$$

$$\nabla \cdot \boldsymbol{u} = 0, \quad \text{in } \Omega, \tag{2.2}$$

$$\boldsymbol{u} = \boldsymbol{u}^D, \quad \text{on } \Gamma, \tag{2.3}$$

where $\boldsymbol{u} = (u_1, \ldots, u_d) \in \mathbb{R}^d$ is the velocity vector, p, the pressure, and $\boldsymbol{f} = (f_1, \ldots, f_d) \in \mathbb{R}^d$, the field of external forces. In equation (2.1), $\alpha \geqslant 0$ is an arbitrary constant. If $\alpha = 0$, then equations (2.1)–(2.3) turn to be the classic Stokes problem. If $\alpha > 0$, then equations (2.1)–(2.3) turn to be a generalized Stokes problem encountered in time discretization of Navier-Stokes equations (see e.g. Glowinski (2003)). The constant $\nu > 0$ is the kinematic viscosity.

We need the functional spaces $\boldsymbol{V} = H_0^1(\Omega)^d$,

$$\boldsymbol{V}^D = \left\{ \boldsymbol{v} \in H^1(\Omega)^d \,:\, \boldsymbol{v} = \boldsymbol{u}^D \text{ on } \Gamma \right\}, \quad P = \left\{ q \in L^2(\Omega) \,:\, \int_\Omega q \, \mathrm{d}x = 0 \right\},$$

and bilinear forms

$$a_i(u_i, v_i) = \alpha (u_i, v_i)_\Omega + \nu (\nabla u_i, \nabla v_i)_\Omega, \quad i = 1, \ldots, d,$$

$$\boldsymbol{a}(\boldsymbol{u}, \boldsymbol{v}) = \sum_{i=1}^{d} a_i(u_i, v_i),$$

where $(\cdot, \cdot)_\Omega$ stands for the standard $L^2(\Omega)$ scalar product. The variational formulation of the Stokes problem (2.1)–(2.3) is as follows:

Find $(\boldsymbol{u}, p) \in \boldsymbol{V}^D \times P$ such that:

$$a(\boldsymbol{u}, \boldsymbol{v}) - (p, \nabla \cdot \boldsymbol{v})_\Omega = (\boldsymbol{f}, \boldsymbol{v})_\Omega, \quad \forall \boldsymbol{v} \in \boldsymbol{V}, \tag{2.4}$$

$$-(q, \nabla \cdot \boldsymbol{u})_\Omega = 0, \quad \forall q \in P. \tag{2.5}$$

3. Finite Element Discretization with $P1$-Bubble-$P1$

For the finite element discretization of (2.4)–(2.5), we have to chose a finite element pair for the velocity field and the pressure. This choice cannot be arbitrary but must satisfy the *inf-sup* condition (Babuska, 1971; Brezzi, 1974).

3.1. *Mini Element*

In this paper we study the discretization of the Stokes problem (2.1)–(2.3) by the finite element pair $P1$-bubble/$P1$ (the so-called *mini-element*), introduced by Arnold *et al.* (1984). This element leads to a relatively low number of degrees of freedom with a good approximate solution.

Let $\mathcal{T}_h$ be a triangulation of Ω and T a triangle of $\mathcal{T}_h$. We define the space associated with the bubble by

$$\mathbb{B}_h = \left\{ \boldsymbol{v}_h \in \mathcal{C}^0(\bar{\Omega});\ \forall T \in \mathcal{T}_h,\ v_{h|T} = \boldsymbol{x} b^{(T)} \right\}.$$

We also defined the discrete function spaces

$$V_{ih} = \left\{ \boldsymbol{v}_h \in \mathcal{C}^0(\bar{\Omega});\ \boldsymbol{v}_{h|T} \in P^1,\ \forall T \in \mathcal{T}_h;\ \boldsymbol{v}_{h|\Gamma} = 0 \right\}, \quad i = 1, \ldots, d,$$

$$P_h = \left\{ q_h \in \mathcal{C}^0(\bar{\Omega});\ q_{h|T} \in P^1,\ \forall T \in \mathcal{T}_h; : \int_\Omega q_h \, \mathrm{d}x = 0 \right\},$$

and we set $X_{ih} = V_{ih} \oplus \mathbb{B}_h$ and $\boldsymbol{X}_h = X_{1h} \times \cdots \times X_{dh}$. With the above preparations, the discrete variational problem reads as follows.

Find $(\boldsymbol{u}_h, p_h) \in \boldsymbol{X}_h \times P_h$ such that

$$\boldsymbol{a}(\boldsymbol{u}_h, \boldsymbol{v}_h) - (p_h, \nabla \cdot \boldsymbol{v}_h)_\Omega = (\boldsymbol{f}, \boldsymbol{v}_h)_\Omega \quad \forall v_h \in \boldsymbol{X}_h, \tag{3.1}$$

$$-(q_h, \nabla \cdot \boldsymbol{u}_h)_\Omega = 0 \quad \forall q_h \in P_h. \tag{3.2}$$

For a given triangle T, the velocity field $\boldsymbol{u}_h$ and the pressure p_h are approximated by linear combinations of the basis functions in the form

$$\boldsymbol{u}_h(\boldsymbol{x}) = \sum_{i=1}^{d+1} \phi_i(\boldsymbol{x})\mathbf{u}_i + \mathbf{u}_b\phi_b(\boldsymbol{x}), \qquad p_h(\boldsymbol{x}) = \sum_{i=1}^{d+1} \phi_i(\boldsymbol{x})\mathrm{p}_i,$$

where $\mathbf{u}_i$ and p_i are nodal values of $\boldsymbol{u}_h$ and p_h while $\mathbf{u}_b$ is the bubble value. The basis functions are defined by

$$\phi_1(\boldsymbol{x}) = 1 - x - y, \qquad \phi_2(\boldsymbol{x}) = x, \qquad \phi_3(\boldsymbol{x}) = y, \qquad \phi_b(\boldsymbol{x}) = 27\prod_{i=1}^{3}\phi_i(\boldsymbol{x})$$

in 2D, and

$$\phi_1(\boldsymbol{x}) = 1 - x - y - z, \qquad \phi_2(\boldsymbol{x}) = x, \qquad \phi_3(\boldsymbol{x}) = y,$$
$$\phi_4(\boldsymbol{x}) = z, \qquad \phi_b(\boldsymbol{x}) = 256\prod_{i=1}^{4}\phi_i(\boldsymbol{x})$$

in 3D.

To construct the coefficient matrices, a number of integrals involving powers of the basis functions will be computed. Integrals over a triangle T can be evaluated directly by the following formulas

$$\int_T \phi_1^{\alpha_1}\phi_2^{\alpha_2}\phi_3^{\alpha_3}\mathrm{d}\boldsymbol{x} = 2|T|\frac{\alpha_1!\alpha_2!\alpha_3!}{(\alpha_1+\alpha_2+\alpha_3+2)!}, \tag{3.3}$$

$$\int_T \phi_1^{\alpha_1}\phi_2^{\alpha_2}\phi_3^{\alpha_3}\phi_4^{\alpha_4}\mathrm{d}\boldsymbol{x} = 6|T|\frac{\alpha_1!\alpha_2!\alpha_3!\alpha_4}{(\alpha_1+\alpha_2+\alpha_3+\alpha_4+3)!}, \tag{3.4}$$

where $|T|$ stands for the triangle area (in 2D), or the tetrahedron volume (in 3D). A useful property for the basis functions is

$$\sum_{i=1}^{d+1}\nabla\phi_i = 0. \tag{3.5}$$

3.2. *Algebraic Formulation*

We use the following notations for the discrete velocity/pressure nodal values

$$\bar{u}_i = \begin{bmatrix} u_i \\ u_{ib} \end{bmatrix}, \qquad \bar{f}_i = \begin{bmatrix} f_i \\ f_{ib} \end{bmatrix}, \quad i = 1, \ldots, d. \tag{3.6}$$

Systems (3.1)–(3.2) lead to the following algebraic form, using notations (3.6)

$$\begin{bmatrix} \bar{A} & 0 & -\bar{B}_1^\top \\ 0 & \bar{A} & -\bar{B}_2^\top \\ -\bar{B}_1 & -\bar{B}_2 & 0 \end{bmatrix} \begin{bmatrix} \bar{u}_1 \\ \bar{u}_2 \\ \mathrm{p} \end{bmatrix} = \begin{bmatrix} \bar{f}_1 \\ \bar{f}_2 \\ 0 \end{bmatrix} \tag{3.7}$$

in 2D, or

$$\begin{bmatrix} \bar{A} & 0 & 0 & -\bar{B}_1^\top \\ 0 & \bar{A} & 0 & -\bar{B}_2^\top \\ 0 & 0 & \bar{A} & -\bar{B}_3^\top \\ -\bar{B}_1 & -\bar{B}_2 & -\bar{B}_3 & 0 \end{bmatrix} \begin{bmatrix} \bar{u}_1 \\ \bar{u}_2 \\ \bar{u}_3 \\ p \end{bmatrix} = \begin{bmatrix} \bar{f}_1 \\ \bar{f}_2 \\ \bar{f}_3 \\ 0 \end{bmatrix} \tag{3.8}$$

in 3D. In (3.7) and (3.8) $\bar{A} = \bar{M} + \bar{R}$, with $\bar{M}$ as the mass matrix and $\bar{R}$ as the stiffness matrix. $\bar{B}_i$ is the divergence submatrix associated with the i-th partial derivative, i.e.

$$\bar{B}_i \equiv (q_h, \partial_i u_{ih})_\Omega, \quad i = 1, \ldots, d.$$

To create the algebraic system (3.7) or (3.8), the discrete system (3.1)–(3.2) is evaluated over each triangle T to obtain the element matrices and vectors

$$\bar{M}_{ij}^{(T)} = \int_T \alpha \phi_i \phi_j \mathrm{d}x, \qquad \bar{R}_{ij}^{(T)} = \int_T \nu \nabla \phi_i \cdot \nabla \phi_j \mathrm{d}x,$$
$$\bar{B}_{ij}^{(T)} = \int_T \partial_1 \phi_i \phi_j \mathrm{d}x + \int_T \partial_2 \phi_i \phi_j \mathrm{d}x, \qquad \bar{f}_i^{(T)} = \int_T f \phi_i \mathrm{d}x.$$

Then assembling operations consist of direct-summing the element matrices over the triangulation $\mathcal{T}_h$ to obtain the global matrices $\bar{M} = (\bar{M}_{ij})$, $\bar{R} = (\bar{R}_{ij})$, $\bar{B} = (\bar{B}_{ij})$ and $\bar{f} = (\bar{f}_i)$

$$\bar{M}_{ij} = \sum_{T \in \mathcal{T}_h} \bar{M}_{ij}^{(T)}, \qquad \bar{R}_{ij} = \sum_{T \in \mathcal{T}_h} \bar{R}_{ij}^{(T)},$$
$$\bar{B}_{ij} = \sum_{T \in \mathcal{T}_h} \bar{B}_{ij}^{(T)}, \qquad \bar{f}_i = \sum_{T \in \mathcal{T}_h} \bar{f}_i^{(T)}.$$

In the next sections we detail the element matrices $\bar{M}$, $\bar{R}$, $\bar{B}_i$ and the right-hand side $\bar{f}$.

4. Element Matrices

For $P1$ finite element, we need only the element area and the gradient of the basis functions for the element matrices and vectors. To derive vectorized MATLAB codes, we need analytical expressions for *all* element matrices and vectors. This section is devoted to this task. For the sake of the presentation we drop the (T)-superscript introduced in the previous section to distinguish element matrices from global ones.

4.1. *Two-Dimensional Case*

For a triangle T, let $\{(x_i, y_i)\}_{i=1,2,3}$ be the vertices and $\{\phi\}_{i=1,2,3}$ the corresponding basis functions. The gradient of ϕ_i is given by

$$\begin{bmatrix} \nabla\phi_1^t \\ \nabla\phi_2^t \\ \nabla\phi_3^t \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \frac{1}{2|T|} \begin{bmatrix} y_2 - y_3 & x_3 - x_2 \\ y_3 - y_1 & x_1 - x_3 \\ y_1 - y_2 & x_2 - x_1 \end{bmatrix}, \tag{4.1}$$

where $|T|$ is the area of T given by

$$2|T| = \det \begin{bmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{bmatrix} = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1).$$

A nonbubble entry of the element stiffness matrix $\bar{R}$ is given by

$$\bar{R}_{ij} = \int_T \nu \nabla\phi_i \cdot \nabla\phi_j \mathrm{d}x = |T| \nu \nabla\phi_i \cdot \nabla\phi_j, \quad i, j = 1, 2, 3. \tag{4.2}$$

For the bubble entries $\bar{R}_{bj}$, for $j = 1, 2, 3$. A straightforward calculation yields to (using (3.5))

$$\bar{R}_{bj} = \frac{9}{4}|T| \sum_{i=1}^{3} \nabla\phi_i = 0, \quad j = 1, 2, 3.$$

For the diagonal entry corresponding to the bubble (i.e. $i = j = b$) we have

$$\begin{aligned} \bar{R}_{bb} &= \nu \int_T 27^2 \nabla(\phi_1\phi_2\phi_3) \cdot \nabla(\phi_1\phi_2\phi_3) \mathrm{d}x \\ &= \frac{81}{10} \nu |T| \left(|\nabla\phi_1|^2 + |\nabla\phi_2|^2 + |\nabla\phi_3|^2 + \nabla\phi_1\nabla\phi_2 + \nabla\phi_1\nabla\phi_3 + \nabla\phi_2\nabla\phi_3 \right) \\ &= \frac{81}{10} \nu |T| \left(|\nabla\phi_1|^2 + |\nabla\phi_2|^2 + \nabla\phi_1 \cdot \nabla\phi_2 \right) =: \omega_R, \end{aligned} \tag{4.3}$$

using (3.5).

With the above results, the element stiffness matrix is therefore

$$\bar{R} = \begin{pmatrix} R & 0 \\ 0 & \omega_R \end{pmatrix}.$$

As for the stiffness matrix, we set $M = (\bar{M}_{ij})_{i,j=1,\ldots,3}$ as the nonbubble part of the mass matrix. A direct calculation yields

$$M_{ij} = \begin{cases} \frac{\alpha}{6}|T| & \text{if } i = j, \\ \frac{\alpha}{12}|T| & \text{elsewhere.} \end{cases} \tag{4.4}$$

The bubble part of the mass matrix is given by

$$\begin{aligned} \bar{M}_{bj} &= \frac{3\alpha}{20}|T|, \quad j = 1, 2, 3, \\ \bar{M}_{bb} &= \frac{81\alpha}{280}|T| =: \omega_M. \end{aligned} \tag{4.5}$$

The element mass matrix is therefore

$$\bar{M} = \begin{pmatrix} M & z \\ z^\top & \omega_M \end{pmatrix},$$

where

$$z^\top = \frac{3}{20}\alpha|T|\,(1\ 1\ 1)\,.$$

Finally, the element stiffness/mass matrix $\bar{A}$ is

$$\bar{A} = \begin{pmatrix} A & z \\ z^\top & \omega \end{pmatrix},$$

where we have set $A = R + M$ and $\omega = \omega_R + \omega_M$.

A direct integration yields the element divergence matrix $-\bar{B} = [-\bar{B}_1\ -\bar{B}_2]$, where $\bar{B}_i = [B_i\ -B_{ib}]$ with

$$B_i = \frac{|T|}{3}\begin{bmatrix} \partial_i\phi_1 & \partial_i\phi_2 & \partial_i\phi_3 \\ \partial_i\phi_1 & \partial_i\phi_2 & \partial_i\phi_3 \\ \partial_i\phi_1 & \partial_i\phi_2 & \partial_i\phi_3 \end{bmatrix}, \quad i = 1, 2, \tag{4.6}$$

and

$$B_{ib} = \frac{9|T|}{20}\begin{bmatrix} \partial_i\phi_1 \\ \partial_i\phi_2 \\ \partial_i\phi_3 \end{bmatrix}, \quad i = 1, 2. \tag{4.7}$$

The contribution of the right-hand side component f_i, in nonbubble terms, is given by

$$f_i^{(T)} = \frac{|T|}{3} f_{iT}, \quad i = 1, 2, 3, \tag{4.8}$$

where f_{iT} is a mean value of f_i on T. The bubble terms of the right-hand side are

$$f_{ib}^{(T)} = \int_T f_i\phi_b \mathrm{d}x = \frac{9}{20}|T|\,f_{iT}, \quad i = 1, 2. \tag{4.9}$$

With the above element matrices and vectors the 11×11 element system corresponding to (3.7) is

$$\begin{bmatrix} A & z & 0 & 0 & -B_1^\top \\ z^\top & \omega & 0 & 0 & B_{1b}^\top \\ 0 & 0 & A & z & -B_2^\top \\ 0 & 0 & z^\top & \omega & B_{2b}^\top \\ -B_1 & B_{1b} & -B_2 & B_{2b} & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_{1b} \\ u_2 \\ u_{2b} \\ p \end{bmatrix} = \begin{bmatrix} f_1 \\ f_{1b} \\ f_2 \\ f_{2b} \\ 0 \end{bmatrix}.$$

To reveal diagonal blocks, the system above can be rearranged as follows

$$\begin{bmatrix} A & 0 & z & 0 & -B_1^\top \\ 0 & A & 0 & z & -B_2^\top \\ z^\top & 0 & \omega & 0 & B_{1b}^\top \\ 0 & z^\top & 0 & \omega & B_{2b}^\top \\ -B_1 & -B_2 & B_{1b} & B_{2b} & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_{1b} \\ u_{2b} \\ p \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_{1b} \\ f_{2b} \\ 0 \end{bmatrix}. \tag{4.10}$$

We can now eliminate the bubble unknowns u_{1b} and u_{2b} since they correspond to diagonal blocks (the ω blocks). From $(4.10)_3$ and $(4.10)_4$, we deduce that

$$u_{ib} = (f_{ib} - B_{ib}^\top p - z^\top u_i)/\omega, \quad i = 1, 2. \tag{4.11}$$

Substituting (4.11) into $(4.10)_1$, $(4.10)_2$ and $(4.10)_5$ we obtain a linear system in $(u_1\ u_2\ p)^\top$ whose matrix is

$$\begin{bmatrix} A - \omega^{-1} z z^\top & 0 & -B_1^\top - \omega^{-1} z B_{1b}^\top \\ 0 & A - \omega^{-1} z z^\top & -B_2^\top - \omega^{-1} z B_{2b}^\top \\ -B_1 - \omega^{-1} B_{1b} z^\top & -B_2 - \omega^{-1} B_{2b} z^\top & -\omega^{-1}(B_{1b} B_{1b}^\top + B_{2b} B_{2b}^\top) \end{bmatrix} \tag{4.12}$$

and the right-hand side

$$\begin{bmatrix} f_1 - \omega^{-1} z f_{1b} \\ f_2 - \omega^{-1} z f_{2b} \\ -\omega^{-1}(B_{1b} f_{1b} + B_{2b} f_{2b}) \end{bmatrix}. \tag{4.13}$$

4.2. *Three-Dimensional Case*

Let $\{\boldsymbol{x}_i = (x_i, y_i, z_i)\}_{i=1,\ldots,4}$ be the vertices of a tetrahedron T and $\{\phi_i\}_{i=1,\ldots,4}$ the corresponding basis functions. The gradient $\nabla_{\boldsymbol{x}}\phi_i$ over T are given by

$$\begin{bmatrix} \nabla\phi_1^\top \\ \nabla\phi_2^\top \\ \nabla\phi_3^\top \\ \nabla\phi_4^\top \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

An alternative formula for computing $\nabla\phi_i$ is

$$\nabla_{\boldsymbol{x}}\phi_i = J^{-1}\nabla_{\boldsymbol{\xi}}\phi_i(\boldsymbol{\xi}),$$

where J is the Jacobean matrix of the mapping $\boldsymbol{\xi} \mapsto \boldsymbol{x}(\boldsymbol{\xi})$, that is,

$$J = \begin{bmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ x_4 - x_1 & y_4 - y_1 & z_4 - z_1 \end{bmatrix}. \tag{4.14}$$

The volume of tetrahedron T is given, from (4.14), by $|T| = \det(J)/6$.

As for 2D case, the nonbubble entries of the element stiffness matrix are given by

$$\bar{R}_{bj} = \frac{9}{4}|T| \sum_{i=1}^{3} \nabla\phi_i = 0, \quad j = 1, 2, 3, 4, \tag{4.15}$$

while $\bar{R}_{bj} = 0$, for all $j = 1, \ldots, 4$. For the diagonal entry, using (3.4)–(3.5), we obtain

$$\begin{aligned} \bar{R}_{bb} &= \nu \int_T 256^2 \nabla(\phi_1\phi_2\phi_3\phi_4) \cdot \nabla(\phi_1\phi_2\phi_3\phi_4) \mathrm{d}x \\ &= \frac{8192}{845}\nu|T| \left[\sum_{\ell=1}^{3} |\nabla\phi_\ell|^2 + \nabla\phi_1 \cdot \nabla\phi_2 + \nabla\phi_1 \cdot \nabla\phi_3 + \nabla\phi_2 \cdot \nabla\phi_3 \right] \\ &=: \omega_R. \end{aligned} \tag{4.16}$$

For the 3D mass matrix, a straightforward calculation with a linear tetrahedron shows that (for nonbubble entries)

$$\bar{M}_{ij} = \begin{cases} \alpha_T \frac{|T|}{10} & \text{if } i = j, \\ \alpha_T \frac{|T|}{20} & \text{if } i \neq j, \end{cases} \tag{4.17}$$

where α_T is a mean value of α on T. The bubble part of the mass matrix is given by

$$\bar{M}_{bj} = \frac{8\alpha}{105}|T|, \quad j = 1, 2, 3,$$

$$\bar{M}_{bb} = \frac{8192\alpha}{51975}|T| =: \omega_M. \tag{4.18}$$

The element mass matrix is therefore

$$\bar{M} = \begin{pmatrix} M & z \\ z^\top & \omega_M \end{pmatrix},$$

where

$$z^\top = \frac{8\alpha}{105}|T| \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}.$$

If we set $-\bar{B} = [-\bar{B}_1 \ -\bar{B}_2 \ -\bar{B}_3]$ and $\bar{B}_i = [B_i \ -B_{ib}]$, a straightforward calculation yields to

$$B_i = \frac{|T|}{4} \begin{bmatrix} \partial_i\phi_1 & \partial_i\phi_2 & \partial_i\phi_3 & \partial_i\phi_4 \\ \partial_i\phi_1 & \partial_i\phi_2 & \partial_i\phi_3 & \partial_i\phi_4 \\ \partial_i\phi_1 & \partial_i\phi_2 & \partial_i\phi_3 & \partial_i\phi_4 \\ \partial_i\phi_1 & \partial_i\phi_2 & \partial_i\phi_3 & \partial_i\phi_4 \end{bmatrix}, \quad i = 1, \ldots, 4, \tag{4.19}$$

and

$$B_{ib} = \frac{32}{105}|T| \begin{bmatrix} \partial_i\phi_1, \\ \partial_i\phi_2 \\ \partial_i\phi_3 \\ \partial_i\phi_4 \end{bmatrix}. \tag{4.20}$$

The contribution of the right-hand side component f_i, in nonbubble terms, is given by

$$f_i^{(T)} = \frac{|T|}{4} f_{iT}, \quad i = 1, 2, 3, 4, \tag{4.21}$$

where f_{iT} is a mean value of f_i on T. The bubble terms of the right-hand side are

$$f_{ib}^{(T)} = \int_T f_i\phi_b \mathrm{d}x = \frac{32}{105}|T| f_{iT}, \quad i = 1, 2, 3. \tag{4.22}$$

As in 2D, we rearrange the Stokes system and we get

$$u_{ib} = \left(f_{ib} - B_{ib}^\top p - z^\top u_i\right)/\omega, \quad i = 1, \ldots, 4. \tag{4.23}$$

Substituting (4.23) into the rest of the system, we obtain a linear system whose matrix is

$$\begin{bmatrix} A - \omega^{-1} zz^\top & 0 & 0 & -B_1^\top - \omega^{-1} z B_{1b}^\top \\ 0 & A - \omega^{-1} zz^\top & 0 & -B_2^\top - \omega^{-1} z B_{2b}^\top \\ 0 & 0 & A - \omega^{-1} zz^\top & -B_3^\top - \omega^{-1} z B_{3b}^\top \\ -B_1 - \omega^{-1} B_{1b} z^\top & -B_2 - \omega^{-1} B_{2b} z^\top & -B_3 - \omega^{-1} B_{3b} z^\top & -\omega^{-1} \sum_{i=1}^3 B_{ib} B_{ib}^\top \end{bmatrix} \tag{4.24}$$

and the right-hand side

$$\begin{bmatrix} f_1 - \omega^{-1} z f_{1b} \\ f_2 - \omega^{-1} z f_{2b} \\ -\omega^{-1} \sum_{i=1}^{3} B_{ib} B_{ib}^{\top} \end{bmatrix}. \tag{4.25}$$

5. Uzawa Conjugate Gradient Algorithm

We propose in this section a preconditioned conjugate gradient method for solving the Stokes system after the assembly of the element systems (4.12)–(4.13) and (4.24)–(4.25). For a finite element pair of the form P_{k+1}/P_k (e.g. $P2/P1$ or $P1$-iso-$P2/P1$), the preconditioner advocated by Cahouet and Chabard (1988) (see also Fortin and Glowinski, 1983; Glowinski, 2003; Glowinski and Le Tallec, 1989) is efficient and widely used. In the author's knowledge, there is no equivalent preconditioner for the pair $P1$-Bubble/$P1$ (or $P1/P1$ with stabilization). The algorithm presented in his section is therefore an original contribution.

The Stokes system can be rewritten in a compact form

$$\begin{bmatrix} \mathbf{A} & -\mathbf{B}^{\top} \\ -\mathbf{B} & -\mathrm{C} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathrm{p} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathrm{f}_p \end{bmatrix}, \tag{5.1}$$

where $\mathbf{A}$ is (symmetric) positive definite block diagonal matrix, C is positive semi-definite matrix.

Let us introduce the generalized Lagrangian function

$$\mathcal{L}(\mathbf{u}, \mathbf{p}) = \frac{1}{2}\mathbf{u}^{\top}\mathbf{A}\mathbf{u} - \mathbf{f}^{\top}\mathbf{u} - \mathrm{p}^{\top}\mathbf{B}\mathbf{u} - \frac{1}{2}\mathrm{p}^{\top}\mathrm{C}\mathrm{p} - \mathrm{f}_p^{\top}\mathrm{p}. \tag{5.2}$$

Due to the properties of $\mathbf{A}$ and $\mathbf{C}$, the saddle-point for (5.2) exists. Then it follows that (5.1) is the saddle-point equation for (5.2), that is, (5.1) characterizes the solution of the saddle-point problem

$$\min_{\mathbf{u}} \max_{\mathrm{p}} \mathcal{L}(\mathbf{u}, \mathrm{p}) = \max_{\mathrm{p}} \min_{\mathbf{u}} \mathcal{L}(\mathbf{u}, \mathrm{p}). \tag{5.3}$$

5.1. *Uzawa Conjuage Gradient Algoritm*

To derive a dual (Uzawa) algorithm for (5.1), we assume that $\mathbf{u} = \mathbf{u}(\mathrm{p})$ is the solution of Poisson equation

$$\mathbf{A}\mathbf{u} = \mathbf{f} + \mathbf{B}^{\top}\mathrm{p}, \tag{5.4}$$

that is, in the decomposed useful form

$$\mathrm{A}_i \mathrm{u}_i = \mathrm{f}_i + \mathrm{B}_i^{\top}\mathrm{p}, \quad i = 1, \ldots, d.$$

Then multiplying (5.4) by $\mathbf{u}$ and substituting the result in (5.2), we obtain

$$\mathcal{L}(\mathbf{u}(\mathrm{p}), \mathrm{p}) = -\frac{1}{2}\mathbf{u}^\top \mathbf{A}\mathbf{u} - \frac{1}{2}\mathrm{p}^\top \mathrm{C}\mathrm{p} - \mathrm{f}_p^\top \mathrm{p}. \tag{5.5}$$

If we introduce the dual functional

$$J^*(\mathrm{p}) = -\min_{\mathbf{u}} \mathcal{L}(\mathbf{u}(\mathrm{p}), \mathrm{p})$$

with $\mathbf{u}(\mathrm{p})$ solution of (5.4), the saddle-point problem (5.3) becomes.

Find p such that

$$J^*(\mathrm{p}) \leqslant J^*(\mathrm{q}), \quad \forall q. \tag{5.6}$$

From (5.5), J^* is quadratic and coercive. From (5.4), we deduce that the mapping $\mathrm{p} \mapsto \mathbf{u}(\mathrm{p})$ is linear and $\mathbf{u}(\mathrm{p}+t\mathrm{d}) = \mathbf{u}(\mathrm{p}) + t\mathbf{w}$ where $\mathbf{w}$ is the solution of the sensitivity problem

$$\mathbf{A}\mathbf{w} = \mathbf{B}^\top \mathrm{d}, \tag{5.7}$$

or

$$\mathrm{A}_i \mathrm{w}_i = \mathrm{B}_i^\top \mathrm{d}, \quad i = 1, \ldots, d.$$

It follows that the derivative of J^* is

$$\mathrm{r} := \nabla J^*(\mathrm{p}) = \mathbf{B}\mathbf{u} + \mathrm{C}\mathrm{p} + \mathrm{f}_p. \tag{5.8}$$

With a search direction d, we compute an optimal stepsize ρ by solving

$$\nabla J^*(\mathrm{p} + \rho \mathrm{d})^\top \mathrm{d} = 0,$$

that is,

$$\rho = -\mathrm{d}^\top (\mathbf{B}\mathbf{w} + \mathrm{C}\mathrm{d})/(\mathrm{r}^\top \mathrm{d}),$$

where $\mathbf{w}$ is the solution of the sensitivity equation (5.7).

Since J^* is a quadratic function, the best search direction is a conjugate gradient direction. As a consequence, the best algorithm for (5.6) is a conjugate gradient algorithm. At each iteration k, the Fletcher–Reeves conjugate gradient direction is given by

$$\mathrm{d}_k = \mathrm{r}_{k+1} + \beta_k \mathrm{d}_k,$$
$$\beta_k = \|\mathrm{r}_{k+1}\|^2 \|\mathrm{r}_k\|^{-2}.$$

A dual (Uzawa) conjugate gradient algorithm for solving the saddle-point problem (5.1) is outlined in Algorithm 1. Theoretically, Algorithm 1 converges in at most $n_B = \operatorname{rank}(\mathbf{B})$ iterations. Obviously, for large scale problems, n_B is very large and it is preferable that convergence should be obtained in a number of iterations considerably less than n_B.

Algorithm 1 Uzawa conjugate gradient algorithm for (5.1).

$k = 0$. p^0 is given

0.1 Initial solution

$$\mathrm{A}_i \mathrm{u}_i^0 = \mathrm{f}_i - \mathrm{B}_i^\top \mathrm{p}_0,\ i = 1, \ldots, d$$

$$\mathrm{r}_0 = \mathbf{B}\mathbf{u}^0 + \mathrm{C}\mathrm{p}_0 + \mathrm{f}_p$$

0.2 Initial gradient and direction

$$\mathrm{r}_0 = \mathbf{B}\mathbf{u}^0 + \mathrm{C}\mathrm{p}_0 + \mathrm{f}_p$$

$$\mathrm{d}_0 = \mathrm{r}_0.$$

$k \geqslant 0$. While $\mathrm{g}_k^\top \mathrm{g}_k > \varepsilon(\mathrm{g}_0^\top \mathrm{g}_0)$

k.1 Sensitivity and stepsize

$$\mathrm{A}_i \mathrm{w}_i^k = \mathrm{B}_i^\top d_k,\ i = 1, \ldots, d$$

$$\tilde{\mathrm{r}}_k = \mathbf{B}\mathbf{w}^k + \mathrm{C}\mathrm{d}_k$$

$$\rho_k = (\mathrm{r}_k^\top \mathrm{d}_k) \,/\, (\mathrm{d}_k^\top \tilde{\mathrm{r}}_k)$$

k.2 Update

$$\mathrm{p}_{k+1} = \mathrm{p}_k - \rho_k \mathrm{d}_k,\ \mathbf{u}^{k+1} = \mathbf{u}^k - \rho_k \mathbf{w}^k$$

k.3 New gradient

$$\mathrm{r}_{k+1} = \mathrm{r}_k - \rho_k \tilde{\mathrm{r}}_k,$$

k.4 Conjugate gradient direction

$$\beta_k = (\mathrm{r}_{k+1}^\top \mathrm{r}_{k+1})(\mathrm{r}_k^\top \mathrm{r}_k)^{-1}$$

$$\mathrm{d}_{k+1} = \mathrm{r}_{k+1} + \beta_k \mathrm{d}_k.$$

5.2. *Preconditioned Uzawa Conjugate Gradient Algorithm*

A practical implementation of a conjugate gradient method for solving (5.1) requires a preconditioner, that is, a suitable scalar product for computing $\nabla J^*(\mathrm{p})$ instead of the standard one used in (5.8). The convergence properties of the conjugate gradient method for the generalized Stokes problem are deteriorated for large values of the ratio α/ν, see e.g. (Glowinski, 2003; Glowinski and Le Tallec, 1989).

To derive a preconditioner for (5.1) following the idea of Cahouet and Chabard (1988), Glowinski and Guidoboni (2009), we need to simplify the continuous problem. We first notice that the equivalence between $P1$-Bubble/$P1$ element and the stabilized formulation has been proved (Pierre, 1987, 1989, 1995; Matsumoto, 2005; Baiocchi *et al.*, 1993). Then, if we neglect the bubble contribution in the stiffness and divergence matrices, (5.1) can be expressed in the strong form as

$$\alpha \boldsymbol{u} - \nu \Delta \boldsymbol{u} + \nabla p = f, \tag{5.9}$$

$$\nabla \cdot \boldsymbol{u} - \nabla \cdot (\nu_h \nabla \boldsymbol{u}) = 0, \tag{5.10}$$

where ν_h is an element dependent constant.

As in Glowinski and Guidoboni (2009) we define the linear operator from $L^2(\Omega)$ into $L^2(\Omega)$ (neglecting the constant term in (5.8))

$$\phi := \mathcal{A}q = \nabla \cdot \boldsymbol{u}_q - \nabla \cdot (\nu_h \nabla q), \tag{5.11}$$

where $\boldsymbol{u}_q$ is the solution of

$$\alpha \boldsymbol{u}_q - \nu \Delta \boldsymbol{u}_q = -\nabla q. \tag{5.12}$$

Note that (5.12) is the (strong) sensitivity system. The idea behind preconditioning is to find a linear operator $\mathcal{B}$ such that $\mathcal{B}\mathcal{A}q = q$. Applying the divergence operator in (5.12), we obtain

$$\alpha \nabla \cdot \boldsymbol{u}_q - \nu \nabla \cdot \Delta \boldsymbol{u}_q = -\Delta q$$

or

$$\alpha \nabla \cdot \boldsymbol{u}_q - \nu \Delta (\nabla \cdot \boldsymbol{u}_q) = -\Delta q. \tag{5.13}$$

Using (5.11) in (5.13), we obtain

$$-\Delta q - \alpha \nabla \cdot (\nu_h \nabla q) + \nu \Delta \big(\nabla \cdot (\nu_h \nabla q)\big) = \alpha \phi - \nu \Delta \phi. \tag{5.14}$$

Then, in practice, at each step of the preconditioned conjugate gradient algorithm, the gradient of J^* is computed as an approximate solution of the linear system

$$(K + \alpha C + \nu K M^{-1} C)\mathrm{g} = (\alpha M + \nu K)\mathrm{r}, \tag{5.15}$$

where K and M are the stiffness and the mass matrices, respectively. Let us introduce the mesh Reynolds number

$$Re_h = \frac{\alpha}{\nu} h^2,$$

where h is the mesh size. Taking into account the CPU time and the storage requirement for computing the last term of the matrix involved in (5.15), we consider the following preconditioning system instead

$$(K + \alpha C)\mathrm{g} = \mathrm{Hr}, \tag{5.16}$$

where

$$\mathrm{H} = \begin{cases} \nu K + \alpha \mathrm{diag}(M) & \text{if } Re_h \leqslant 1, \\ \nu K + \alpha \mathbb{I} & \text{if } Re_h > 1. \end{cases} \tag{5.17}$$

Algorithm 2 Preconditioned Uzawa conjugate gradient algorithm for (5.1).

$k = 0$. p^0 is given: Set $\mathrm{P} = K + \alpha C$, H given by (5.17).

0.1 Initial solution and residual

$$\mathrm{A}_i \mathrm{u}_i^0 = \mathrm{f}_i - \mathrm{B}_i^\top \mathrm{p}_0,\ i = 1, \ldots, d$$

$$\mathrm{r}_0 = \mathbf{B}\mathbf{u}^0 + \mathrm{C}\mathrm{p}_0 + \mathrm{f}_p$$

0.2 Initial gradient and direction

$$\mathrm{P}\mathrm{g}_0 = \mathrm{H}\mathrm{r}_0$$

$$\mathrm{d}_0 = \mathrm{g}_0.$$

$k \geqslant 0$. While $\mathrm{g}_k^\top \mathrm{r}_k > \varepsilon(\mathrm{g}_0^\top \mathrm{r}_0)$

k.1 Sensitivity and stepsize

$$\mathrm{A}_i \mathrm{w}_i^k = \mathrm{B}_i^\top d_k,\ i = 1, \ldots, d$$

$$\tilde{\mathrm{r}}_k = \mathbf{B}\mathbf{w}^k + \mathrm{C}\mathrm{d}_k$$

$$\rho_k = (\mathrm{r}_k^\top \mathrm{d}_k) \,/\, (\mathrm{d}_k^\top \tilde{\mathrm{r}}_k)$$

k.2 Update

$$\mathrm{p}_{k+1} = \mathrm{p}_k - \rho_k \mathrm{d}_k,\ \mathrm{r}_{k+1} = \mathrm{r}_k - \rho_k \tilde{\mathrm{r}}_k,$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k - \rho_k \mathbf{w}^k$$

k.3 New gradient

$$\mathrm{P}\tilde{\mathrm{g}}_k = \mathrm{H}\tilde{\mathrm{r}}_k,$$

$$\mathrm{g}_{k+1} = \mathrm{g}_k - \rho_k \tilde{\mathrm{g}}_k$$

k.4 Conjugate gradient direction

$$\beta_k = (\mathrm{g}_{k+1}^\top \mathrm{r}_{k+1})(\mathrm{g}_k^\top \mathrm{r}_k)^{-1}$$

$$\mathrm{d}_{k+1} = \mathrm{g}_{k+1} + \beta_k \mathrm{d}_k.$$

In (5.16) and (5.17), K and M are $P1$ stiffness and mass matrix, respectively, and C, the bubble matrix from Functions 6.1–6.3.

From the theory of preconditioned conjugate gradient methods for the Stokes problem (see e.g. Glowinski, 2003), if Dirichlet conditions are imposed for the velocity, then for g in (5.14) we must impose $\partial \mathrm{g}/\partial n = 0$ (homogeneous Neumann boundary conditions). On the other hand, where a stress condition is prescribed for the fluid, we must impose $\mathrm{g} = 0$ in (5.14).

REMARK 1. The preconditioning system (5.15), that is,

$$(\alpha M + \nu K)^{-1}(K + \alpha C + \nu K C)\mathrm{g} = \mathrm{r},$$

induces (over the discrete pressure space) the norm $|\mathrm{g}|_P = \mathrm{g}^\top \mathrm{r}$.

REMARK 2. If $C = 0$ in (5.15), we recover the preconditioning system of Cahouet and Chabard (1988) widely used for $P2/P1$ or $P1$-iso-$P2/P1$ finite elements.

With the preparation given in the previous section, the preconditioned Uzawa conjugate gradient algorithm for solving the Stokes system (5.1) is detailed in Algorithm 2.

Algorithm 2 inherits the main properties from the conjugate gradient algorithm of Cahouet and Chabard (1988):

- At each step, we solve d independent linear systems for the velocity field. The matrices involved are sparse and can be factorized or preconditioned once and for all.
- The same algorithm is able to deal with 2D or 3D systems without any complication.

6. MATLAB Implementation

We know in detail the assembly of the Stokes systems (4.12)–(4.13) and (4.24)–(4.25). For the computational efficiency, the MATLAB codes must be vectorized (i.e. without long *for*-loops). We then use arrays, cell-arrays, and MATLAB vectorized operators and functions

```
.*,   ./,  .^,  sum, sparse
```

6.1. *Mesh Representation and KPDE Package*

We assume that the triangulation of Ω consists of `np` nodes and `nt` elements (triangles or tetrahedrons). We adopt the mesh representation by arrays used in Koko (2007, 2016), Persson and Strang (2004). The nodes coordinates are stored in an array `p(1:np,1:2)` (in 2D) or `p(1:np,1:3)` (in 3D). The element nodes are stores in an array `t(1:nt,1:3)` (in 2D) or `t(1:nt,1:4)` (in 3D). Dirichlet boundary conditions are provided by a list of nodes and the corresponding prescribed values.

As shown in Koko (2016), using cell-array in FEM allows to have implementation close to the standard form used in classical languages (C/C++, FORTRAN) FEM codes, while being efficient. The idea is to compute and store, for all triangles, the element matrix entry $A_{ij}^{(T)}$ in the cell-array `At{i,j}`. Then we use MATLAB function `sparse` to assemble A with finite small *for* loops

```
for i=1:nd
for j=1:nd
   A=A+sparse(t(:,i),t(:,j),At{i,j},np,np);
end
end
```

where `nd=3` (in 2D) or `nd=4` (in 3D). This approach is used in KPDE package (Koko, 2016) for assembling matrices and vectors from Poisson and linear elasticity equations in 2D and 3D.

We need two key functions from KPDE package, `kpde2dgphi` and `kpde3dgphi`, which compute the gradient of the basis functions and the elements volume as follows

```
[ar,g]=kpde2dgphi(p,t);      %2D
[vol,g]=kpde3dgphi(p,t);     %3D
```

`g` is 3-by-1 or 4-by-1 cell-array such that `g{i}(:,k)` is $\partial_k \phi_i(x)$ for all elements.

6.2. *2D Case*

Using the cell array `g` and the array `ar`, the nonbubble entry $\bar{R}_{ij}$, (4.2), of the element stiffness matrix is then given (for all triangles) by

```
Rij=nu*ar.*sum(g{i}.*g{j},2).
```

The bubble diagonal entry ω_R, (4.3), is computed as follows for all triangles

```
omega_r=(81/10)*nu.*ar.*(sum(g1.^2,2)+sum(g2.^2,2)
         +sum(g1.*g2,2)).
```

The nonbubble entry M_{ij}, (4.4), of the element mass matrix is,

```
Mii=alpha.*ar/6 ;        % diagonal
Mij=alpha.*ar/12;        % off-diagonal.
```

The bubble entries (4.5) of the element mass matrix are given by

```
omega_m=(81/280)*alpha.*ar;     % diagonal
Mbj=(3/20)*alpha.*ar};          % off-diagonal.
```

The entries (i, j) of the element divergence matrix (4.6) are vectorized as follows

```
B1ij=ar.*g{i}(:,1)/3; B2ij=ar.*g{i}(:,2)/3;
```

while the bubble entries (4.7) are

```
B1ib=(9/10)*ar.*g{i}(:,1); B2ib=(9/10)*ar.*g{i}(:,2).
```

The vectorized element contribution of the right-hand side (4.8) is

```
f1=(1/3)*ar.*f1t; f2=(1/3)*ar.*f2t;
```

while the bubble entries (4.9) are

```
f1b=(9/20)*ar.*f1t; f2b=(9/20)*ar.*f2t.
```

MATLAB Functions 6.1–6.2 assemble the Stokes system. To make the assembling functions self-contained, all calculations are integrated except elements area and the gradient of basis functions which can be computed outside and passed as argument. Note that in 2D, the time for computing the triangles area and the gradient of the basis functions is not significant. Functions 6.1–6.2 can be called without the last two arguments without a significant computational overcost.

Instead of matrix (4.12) and vector (4.13), Functions 6.1–6.2 can return submatrices and subvectors of (4.12)–(4.13) if called with more than one output arguments. The statement

```
[A,B,C]=kstok2dp1bmat(p,t,nu,alpha)
```

returns the velocity stiffness matrix `A`, the divergence matrix `B=[B1 B2]` and the pressure (stiffness) matrix `C`. Similarly,

```
[b,bp]=kstok2dp1brhs(p,t,nu,alpha)
```

returns the velocity right-hand side `b=[b1; b2]` and the pressure right-hand side `bp`. These submatrices are used in the preconditioned Uzawa conjugate gradient method presented in Section 5.

Function 6.1 Assembly of the 2D Stokes matrix (4.12).

```
function [A,B,C]=kstok2dp1bmat(p,t,nu,alpha,ar,g)
% A=kstok2dp1bmat(p,t,nu,alpha) or [A,B,C]=kstok2dp1bmat(p,t,nu,alpha)
% A=kstok2dp1bmat(p,t,nu,alpha,ar,g) or [A,B,C]=kstok2dp1bmat(p,t,nu,alpha,ar,g)
%--------------------------------------------------------------------------
np=size(p,1); Z=sparse(np,np);
% Triangles area and gradient of basis functions
if (nargin == 4) [ar,g]=kpde2dgphi(p,t);  end
% Bubble coefficients (z and omega)
zt=(3/20)*alpha.*ar;
omega=(81/10)*nu.*ar.*(sum(g{1}.^2,2)+sum(g{2}.^2,2)+sum(g{1}.*g{2},2))...
     +(81/280)*alpha.*ar;
c=(9/10)*ar;
% Matrices assembly
Ah=Z;  Bh=cell(1,2); [Bh{:}]=deal(Z); Ch=sparse(np,np);
for i=1:3
    for j=1:3
       Ah=Ah+sparse(t(:,i),t(:,j),nu.*ar.*sum(g{i}.*g{j},2),np,np)...
            +sparse(t(:,i),t(:,j),alpha.*ar/12,np,np)...
            -sparse(t(:,i),t(:,j),zt.*zt./omega,np,np);
       Ch=Ch-sparse(t(:,i),t(:,j),c.*c.*sum(g{i}.*g{j},2)./omega,np,np);
       for k=1:2
           Bh{k}=Bh{k}-sparse(t(:,i),t(:,j),ar.*g{j}(:,k)/3,np,np)
                      -sparse(t(:,i),t(:,j),c.*g{i}(:,k).*zt./omega,np,np);
       end
    end
    Ah=Ah+sparse(t(:,i),t(:,i),alpha.*ar/12,np,np);
end
% Output
if (nargout == 1)  A=[Ah   Z   Bh{1}'; Z    Ah  Bh{2}'; Bh{1}   Bh{2}  Ch];
elseif (nargout > 1) A=Ah; B=[-Bh{1} -Bh{2}]; C=-Ch; end
```

Function 6.2 Assembly of the Stokes right-hand side (4.13).

```
function [b,bp]=kstok2dp1brhs(p,t,f1,f2,nu,alpha,ar,g)
% b=kstok2dp1brhs(p,t,f1,f2,nu,alpha) or [b,bp]=kstok2dp1brhs(p,t,f1,f2,nu,alpha)
% b=kstok2dp1brhs(p,t,f1,f2,nu,alpha,ar,g) or
% [b,bp]=kstok2dp1brhs(p,t,f1,f2,nu,alpha,ar,g)
%--------------------------------------------------------------------------
np=size(p,1);
% (f1,f2) at the center of triangles
if (length(f1)==np), f1t=sum(f1(t),2)/3;  else f1t=f1; end
if (length(f2)==np), f2t=sum(f2(t),2)/3;  else f2t=f2; end
% Triangles area and gradient of basis functions
if (nargin == 6) [ar,g]=kpde2dgphi(p,t);  end
% Bubble coefficients (z and omega)
zt=(3/20)*alpha.*ar;
omega=(81/10)*nu.*ar.*(sum(g{1}.^2,2)+sum(g{2}.^2,2)+sum(g{1}.*g{2},2))...
     +(81/280)*alpha.*ar;
c=(9/10)*ar; ft={f1t f2t};
% Assembly of the right-hand side
bb=sparse(np,1); bh=cell(2,1); [bh{:}]=deal(sparse(np,1));
for i=1:3
for k=1:2
    bh{k}=bh{k}+sparse(t(:,i),1,(1/3)*ft{k}.*ar-c.*ft{k}.*zt./omega,np,1);
    bb=bb-sparse(t(:,i),1,c.*c.*g{i}(:,k).*ft{k}./omega,np,1);
end
end
% Right-hand side
if (nargout == 1)    b=[full(cell2mat(bh)); full(bb)];
elseif (nargout == 2) b=full(cell2mat(bh)); bp=full(bb); end
```

6.3. *3D Case*

Using the cell-array `g` and the element volume `vol`, computed by the KPDE function `kpde3dgphi`, the element stiffness matrices are computed simultaneously on all elements by

```
Rij=nu*vol.*sum(g{i}.*g{j},2);
```

while bubble diagonal entry (4.16) is given by

```
omega_R=(8192/945)*nu.*vol.*(sum(g{1}.^2,2)+sum(g{2}.^2,2)
        +sum(g{3}.^2,2)...+sum(g{1}.*g{2},2)
        +sum(g{1}.*g{3},2)+sum(g{2}.*g{3},2)).
```

The nonbubble entries of element mass matrix (4.17) are computed by

```
Mii=alpha.*vol/10;      % diagonal
Mij=alpha.*vol/20;      % off diagonal.
```

The bubble part of the mass matrix (4.18) is given by

```
omega_m=(8192/51975)*alpha.*vol;    % diagonal
Mbj=(8/105)*alpha.*vol;             % off-diagonal.
```

The entries (i, j) of the element divergence matrices (4.19) are computed simultaneously as follows

```
B1ij=vol.*g{i}(:,1)/4;  B2ij=vol.*g{i}(:,2)/4;
B3ij=vol.*g{i}(:,3)/4;
```

while the bubble entries are

```
B1ib=(32/105)*vol.*g{i}(:,1); B2ib=(32/105)*vol.*g{i}(:,2);
B3ib=(32/105)*vol.*g{i}(:,3).
```

The element contributions of the right-hand side (4.21)–(4.22) are given by

```
f1=vol.*f1t/4; f2=vol.*f2t/4; f3=vol.*f3t/4;
f1b=(32/105)*vol*f1t;   f2b=(32/105)*vol*f2t;
f3b=(32/105)*vol*f3t.
```

MATLAB Functions 6.3–6.4 assemble the three-dimensional Stokes system. For computational efficiency, the elements volume `vol` and the gradient of the basis functions `g` can be computed once and for all, and passed to Functions 6.3–6.4. For more flexibility, `vol` and `g` can also be computed inside Functions 6.3–6.4 if they do not appear in the list of input arguments. As in 2D, Functions 6.3–6.4 can return submatrices and subvectors used in the preconditioned Uzawa conjugate gradient algorithm, if called with more than one output argument.

Function 6.3 Assembly of the 3D Stokes matrix (4.24).

```
function [A,B,C]=kstok3dp1bmat(p,t,nu,alpha,vol,g)
%-------------------------------------------------------------------------
% A=kstok3dp1bmat(p,t,nu,alpha) or A=kstok3dp1bmat(p,t,nu,alpha,vol,g)
% [A,B,C]=kstok3dp1bmat(p,t,nu,alpha) or [A,B,C]=kstok3dp1bmat(p,t,nu,alpha,vol,g)
%-------------------------------------------------------------------------------
np=size(p,1); Z=sparse(np,np);
% Gradient of basis functions
if (nargin == 4) [vol,g]=kpde3dgphi(p,t); end
% Bubble coefficients ( z and omega )
zt=(8/105)*alpha.*vol;
omega=(8192/945)*nu.*vol.*(sum(g{1}.^2,2)+sum(g{2}.^2,2)+sum(g{3}.^2,2)...
     +sum(g{1}.*g{2},2)+sum(g{1}.*g{3},2)+sum(g{2}.*g{3},2))+(8192/51975)*alpha.*vol;
c=(32/105)*vol;
% Stiffness, mass and divergence matrices
Ah=sparse(np,np); Bh=cell(1,3); [Bh{:}]=deal(Z); Ch=sparse(np,np);
for i=1:4
   for j=1:4
       Ah=Ah+sparse(t(:,i),t(:,j),nu*vol.*sum(g{i}.*g{j},2),np,np)...
            +sparse(t(:,i),t(:,j),alpha*vol/20,np,np)...
            -sparse(t(:,i),t(:,j),zt.*zt./omega,np,np);
       Ch=Ch-sparse(t(:,i),t(:,j),c.*c.*sum(g{i}.*g{j},2)./omega,np,np);
       for k=1:3
           Bh{k}=Bh{k}-sparse(t(:,i),t(:,j),vol.*g{j}(:,k)/4,np,np)...
                      -sparse(t(:,i),t(:,j),c.*g{i}(:,k).*zt./omega,np,np);
       end
   end
   Ah=Ah+sparse(t(:,i),t(:,i),alpha*vol/20,np,np);
end
% Final matrix
if (nargout == 1)
  A=[Ah Z  Z  Bh{1}'; Z  Ah Z  Bh{2}'; Z  Z  Ah Bh{3}'; Bh{1} Bh{2} Bh{3} Ch];
elseif (nargout == 3) A=Ah; B=[-Bh{1} -Bh{2} -Bh{3}]; C=-Ch; end
```

6.4. *Preconditioned Uzawa Conjugate Gradient Algorithm*

In our MATLAB implementation the same function (i.e. `kstokcg`) is used for 2D and 3D problems. For this, we use cell-arrays to store the component system informations: Cholesky factors, permutation vectors, right-hand sides. For instance, for the 3D Stokes problem we form

```
R={R1 R2 R3};   % Cholesky factors
s={s1 s2 s3};    % Permutations vectors
b={b(1:np) b(np+1:2*np) b3(2*np+1:3*np)}; %Right-hand sides.
```

Then in the conjugate gradient function, the velocity systems are solved using the for-loop

```
for i=1:nd
     w{i}(s{i})=R{i}'\(R{i}\b{i}(s{i}));
end
```

where `nd=3`. An alternative implementation is to form the block diagonal matrix `R=blkdiag(R1, R2, R3)`, the permutation vector `s=[s1 s2 s3]` such that e.g. in Step k.1, we solve

```
w(s)=R'\(R\b(s)).
```

Function 6.4 Assembly of the 3D Stokes right-hand side (4.25).

```
function [b,bp]=kstok3dp1brhs(p,t,f1,f2,f3,nu,alpha,vol,g)
% b=kstok3dp1brhs(p,t,f1,f2,f3,nu,alpha) or
% b=kstok3dp1brhs(p,t,f1,f2,f3,nu,alpha,vol,g)
% [b,bp]=kstok2dp1brhs(p,t,f1,f2,nu,alpha) or
% [b,bp]=kstok2dp1brhs(p,t,f1,f2,nu,alpha,vol,g)
%-------------------------------------------------------------------------
np=size(p,1);
% (f1,f2) at the center of triangles
if (length(f1)==np), f1t=sum(f1(t),2)/4;  else f1t=f1; end
if (length(f2)==np), f2t=sum(f2(t),2)/4;  else f2t=f2; end
if (length(f3)==np), f3t=sum(f3(t),2)/4;  else f3t=f3; end
% Triangles area
if (nargin == 7) [vol,g]=kpde3dgphi(p,t); end
% Bubble coefficients (z and omega)
zt=(8/105)*alpha.*vol;
omega=(8192/945)*nu.*vol.*(sum(g{1}.^2,2)+sum(g{2}.^2,2)+sum(g{3}.^2,2)
      +sum(g{1}.*g{2},2)...+sum(g{1}.*g{3},2)+sum(g{2}.*g{3},2))
      +(8192/51975)*alpha.*vol;
c=(32/105)*vol;
% Assembly of the right-hand side
ft={f1t f2t f3t};
bb=sparse(np,1);  bh=cell(3,1); [bh{:}]=deal(sparse(np,1));
for i=1:4
for k=1:3
    bh{k}=bh{k}+sparse(t(:,i),1,(1/4)*ft{k}.*vol-c.*ft{k}.*zt./omega,np,1);
    bb=bb+sparse(t(:,i),1,c.*c.*g{i}(:,k).*ft{k}./omega,np,1);
end
end
% Output
if (nargout == 1)     b=[full(cell2mat(bh)); full(bb)];
elseif (nargout == 2) b=full(cell2mat(bh)); bp=full(bb); end
```

But for large scale 3D problems, computing `R'\(R\b(s))` requires a large amount of memory and can fail.

7. Numerical Experiments

We now propose some numerical experiments to demonstrate the performances of our implementation. The computations have been carried out on a Dell Precision T3610 work station equipped with Intel Xeon 3.0GHz processor with 32GB RAM. The MATLAB version is 9 (R2016a).

7.1. *Scalability*

We first study the scalability of our MATLAB codes: We consider the discretization of a unit cube $(0,\ 1)^d$ $(d = 2, 3)$ with a uniform mesh of size h, with `nt` triangles (or tetrahedrons) and `np` nodes. This initial mesh is successively uniformly refined to produce meshes of size $h/2$, $h/4$, $h/8$, etc. After each refinement the number of triangles is multiplied by 4 (2D) and the number of tetrahedraons by 8 (3D). Since the assembly process is essentially based on the number of elements, we expect that the time to assemble the matrices increases by approximately the same factor, i.e. 5–6 in 2D and 8–10 in 3D as observed in Koko (2016) for Poisson equation and linear elasticity. Tables 1–2 show the

Table 1
CPU times (in seconds) for assembling the Stokes matrix system of size N in 2D.

h	1/32	1/64	1/128	1/256	1/512	1/1024
N	3*1089	3*4225	3*16641	3*66049	3*263169	3*1050625
A	0.018	0.071	0.325	1.516	7.242	36.390
b	0.002	0.011	0.050	0.251	1.214	5.995

Table 2
CPU times (in seconds) for assembling the Stokes matrix system of size N in 3D.

h	1/4	1/8	1/16	1/32	1/64	1/128
N	4*125	4*729	4*4913	4*35937	4*274625	4*2146689
A	0.075	0.057	0.397	3.438	34.538	348.249
b	0.027	0.018	0.051	0.465	4.544	46.394

Table 3
Comparative performances of MATLAB direct solvers and Algorithm 2 for the 2D Stokes system, $\nu = 1/50$ and $\alpha/\nu = 10^3$.

Mesh size h	1/32	1/64	1/128	1/256	1/512
Gaussian elim. CPU	0.02	0.07	0.40	2.11	10.88
LDL$^\top$ CPU	0.02	0.17	1.22	7.13	46.11
Algorithm 2 CPU	0.02	0.08	0.35	1.94	12.01

assembly CPU times (in Seconds) for the Stokes system in 2D and 3D, respectively. We can notice an almost linear optimal time-scaling for our implementation.

7.2. *Factorization Versus Uzawa Conjugate Gradient*

We now consider a Stokes flow in a driven cavity $\Omega = (0,\ 1)^d$ with $\boldsymbol{f} = 0$ in (2.1) and

2D: $\Gamma_1 = (0,\ 1) \times \{1\}$, $\boldsymbol{u} = (1,\ 0)^\top$ on Γ_1, and $\boldsymbol{u} = 0$ on $\partial\Omega \setminus \Gamma_1$;
3D: $\Gamma_1 = (0,\ 1) \times (0,\ 1) \times \{1\}$, $\boldsymbol{u} = (1,\ 0,\ 0)^\top$ on Γ_1, and $\boldsymbol{u} = 0$ on $\partial\Omega \setminus \Gamma_1$.

Ω is discretized by uniform meshes of size $1/16$, $1/32$, $1/64$, $1/128$, $1/256$ and $1/512$ in 2D, and $1/4$, $1/8$, $1/16$, $1/32$ and $1/64$ in 3D. Since there are powerful linear (direct) solvers in MATLAB, we first compare our conjugate gradient algorithm with

- `\` (backslash) the standard MATLAB solver for general matrix based on Gaussian elimination;
- `ldl` the block $LDL^\top$ factorization for symmetric indefinite systems.

Table 3 shows the comparative performances for the 2D Stokes problem with $\nu = 1/50$ and $\alpha/\nu = 10^3$. We can notice that the proposed Uzawa conjugate gradient (Algorithm 2) and the MATLAB built-in Gaussian elimination are almost equivalent even though, in Algorithm 2, the component systems are uncoupled and can be solved in parallel. The CPU times for $LDL^\top$ include CPU time for the factorization and columns and rows permutation to reduce fill-in that represents up to 90% of the whole CPU time.

Table 4
Comparative CPU times (in sec.) of MATLAB direct solvers and Algorithm 2 for the 3D Stokes system, $\nu = 1/50$ and $\alpha/\nu = 10^3$, (OoM = Out of Memory).

Mesh size h	1/4	1/8	1/16	1/32	1/64
Gaussian elim.	0.01	0.05	1.73	145.44	OoM
$\mathrm{LDL}^\top$	0.02	0.08	6.29	128.54	OoM
Algorithm 2	0.04	0.89	1.70	78.08	1894.59

Table 5
Performances of the `gmres` iterative solver for the 3D Stokes system, $\nu = 1/50$ and $\alpha/\nu = 10^3$.

Mesh size h	1/4	1/8	1/16	1/32	1/64
CPU Times (sec.)	0.04	0.02	1.66	39.61	8230.59

Table 6
Number of iterations versus α/ν for the 3D driven cavity problem for $h = 1/32$.

α/ν	10^1	10^2	10^3	10^4	10^5	10^6	10^7
$\nu = 1/50$	44	39	41	28	13	8	7
$\nu = 1/200$	37	34	33	26	13	8	7
$\nu = 1/1000$	31	28	27	24	18	8	7

If the Stokes problem is used in an iterative process (e.g. time stepping or linearization), then Algorithm 2 or $LDL^\top$ factorization are preferable. Indeed, if a $LDL^\top$ is carried out (once and for all) in the initialization step, then the solution of linear system reduces to forward/backward substitutions in the rest of the iterative process. The computational cost of Algorithm 2 can be reduced by using, as initial solution at the current step, the solution of the previous step.

For the 3D Stokes problem the proposed Algorithm 2 outperforms the MATLAB direct solvers, Table 4. For the largest problem ($4 * 65^3 = 1\,098\,500$ unknowns) the direct solvers fail because of lack of memory due to fill-in during factorization. Table 5 shows the performances of the MATLAB `gmres` iterative solver using incomplete LU factorization as preconditioner (MATLAB function `ilu` with 10^{-3} as drop tolerance). The value of restart parameter is 10. GMRES algorithm outperforms Algorithm 2 up to $h = 1/32$. But for the largest problem, Algorithm 2 is more than four times faster. It is clear that for large scale 3D problems, the proposed Uzawa algorithm is preferable. Table 6 shows the good convergence properties of the proposed Uzawa conjugate gradient algorithm when $\alpha/\nu \gg 1$.

7.3. *2D Visualization*

In two-dimensional incompressible fluid problems, it is usual to display the stream-lines. If the domain Ω is bounded and simply connected, in order to compute the stream-function

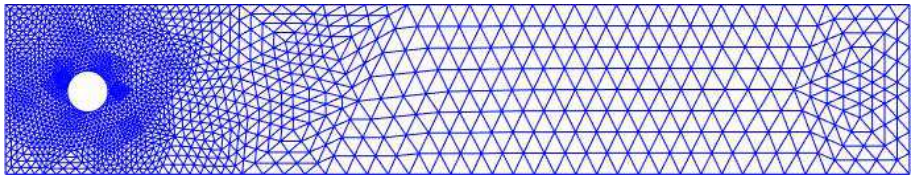

Fig. 1. Mesh sample of the flow around cylinder problem.

ψ, we have to solve the Poisson–Neumann problem

$$-\Delta\psi = \omega, \text{ in } \Omega, \tag{7.1}$$

$$\partial_n\psi = -\boldsymbol{u}\cdot\tau, \tag{7.2}$$

where $\omega = \partial_1 u_2 - \partial_2 u_1$ is the vorticity and τ the counter-clockwise oriented unit tangent vector at Γ. Problem (7.1)–(7.2) has a unique solution in $H^1(\Omega)/\mathbb{R}$. The variational formulation of (7.1)–(7.2) is

Find $\psi \in H^1(\Omega)$

$$(\nabla\psi, \nabla\varphi)_\Omega = (u_1, \partial_2\varphi)_\Omega - (u_2, \partial_1\varphi)_\Omega, \quad \forall\varphi \in H^1(\Omega). \tag{7.3}$$

This leads to the following algebraic system using $P1$ finite element

$$R\psi = B_2^\top \mathrm{u}_1 - B_1^\top \mathrm{u}_2,$$

where R is the 2D Laplacian matrix. We impose $\psi = 0$ at an arbitrary node to ensure the uniqueness.

We now consider a test problem derived from a benchmark problem described in Schäfer and Turek (1996). A mesh sample is shown in Fig. 1. The inflow and outflow conditions (on left/right boundaries) are

$$u_1 = \frac{0.3}{0.41^2}\times 4y(0.41-y),\ u_2 = 0 \quad \text{on } \Gamma_{in} = \{0\}\times(0, 0.41),$$

$$u_1 = \frac{0.3}{0.41^2}\times 4y(0.41-y),\ u_2 = 0 \quad \text{on } \Gamma_{out} = \{2.2\}\times(0, 0.41).$$

On the other parts of the boundary of Ω, homogeneous boundary conditions are prescribed (i.e. $\boldsymbol{u} = 0$). The parameter α in (2.1) is set to 0. The centre of the internal cylinder is $(0.25, 0.2)$ and the diameter is 0.1. The kinematic viscosity is $\nu = 10^{-3}$. This gives a Reynolds number of $Re = 30$ based on the diameter of the cylinder and the maximum of the inflow velocity. The domain is discretized by a non uniform mesh consisting of 1730 nodes and 3280 triangles, Fig. 1. The velocity field obtained with MATLAB command

```
quiver(p(:,1),p(:,2),u1,u2)
```

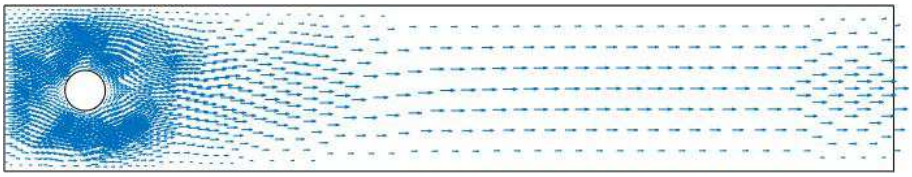

Fig. 2. Velocity field of the flow around cylinder problem.

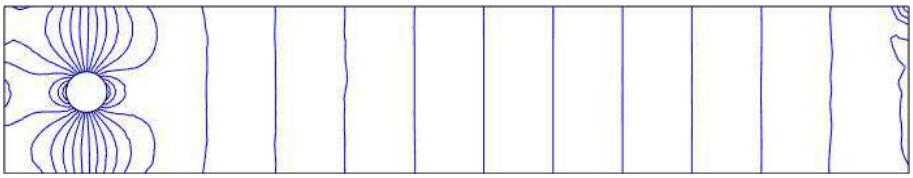

Fig. 3. Isobar lines for the flow around a cylinder problem.

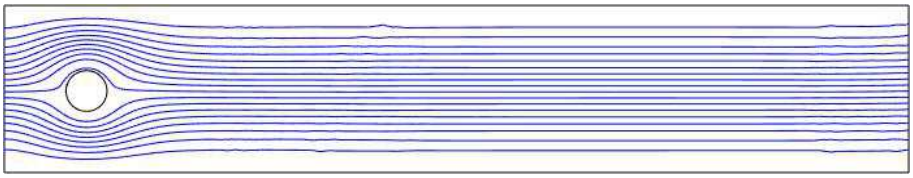

Fig. 4. Streamlines for the flow around a cylinder problem.

is shown in Fig. 2. Isobar lines of Fig. 3 are obtained using the contour plotting function `kpde2dcont` from KPDE package (Koko, 2016). The streamlines of Fig. 4 are obtained by plotting the solution of (7.3) with `kpde2dcont`.

Unfortunately, for 3D flows, there is no simple tool for graphics output. `quiver3` allows for visualization of 3D velocity fields but the result is often unsatisfactory. Plotting 3D functions (or their contours) is a non trivial problem. There is no simple subproblem like (7.3) for streamlines in 3D.

8. Conclusion

We have proposed a fast MATLAB package for the numerical approximation of the generalized Stokes problem with the mini-element. Numerical experiments show that the proposed assembling functions have an optimal linear time-scaling. The proposed Uzawa conjugate gradient algorithm outperforms the MATLAB built-in solvers for 3D problems.

References

Alberty, J., Carstensen, C., Funken, S.A. (1999). Remarks around 50 lines of matlab: short finite element implementation. *Numerical Algorithms*, 20, 117–137.

Alberty, J., Carstensen, C., Funken, S.A., Klose, R. (2002). Matlab implementation of the finite element method in elasticity. *Computing*, 69, 239–263.
Arnold, D., Brezzi, F. Fortin, M., (1984). A stable finite element for the Stokes equations. *Calcolo*, 21, 337–344.
Babuska I. (1971). Error bounds for finite element method. *Numerische Mathematik*, 16, 322–333.
Baiocchi, C., Brezzi, F., Franca, L.P. (1993). Virtual bubbles and Galerkin-Least-Squares methods. *Computer Methods in Applied Mechanics and Engineering*, 105, 125–141.
Brezzi, F. (1974). On the existence, uniqueness and approximation of saddle-point problems arising from Lagrange multipliers. *RAIRO*, 8, 129–151.
Cahouet, J., Chabard, J.P. (1988). Some fast 3-D solvers for the generalized Stokes problem. *International Journal for Numerical Methods in Fluids*, 8, 269–295.
Ern, A., Guermond, J.-L. (2002). *Eléments finis: théorie, applications, mise en œuvre*. *SMAI Mathématiques et Applications*, Vol. 36. Springer.
Fortin, M., Glowinski, R. (1983). *Augmented Lagrangian Methods: Application to the Numerical Solution of Boundary-Value Problems*. North-Holland, Amsterdam.
Glowinski, R. (2003). Numerical methods for fluids (part 3). In: Ciarlet, P.G., Lions, J.L. (Eds.), *Numerical Methods for Fluids (Part 3)*, *Handbook of Numerical Analysis*, Vol. IX. North-Holland, Amsterdam, pp. 3–1074.
Glowinski, R., Guidoboni, G. (2009). On the predoncitioned conjugate gradient solution of a Stokes problem with Robin-type boundary conditions. *Comptes Rendus de l'Académie des Sciences Paris*, 347(15), 903–908.
Glowinski, R., Le Tallec, P. (1989). *Augmented Lagrangian and Operator-splitting Methods in Nonlinear Mechanics*. *Studies in Applied Mathematics*. SIAM, Philadelphia.
Koko J. (2007). Vectorized MATLAB codes for two-dimensional linear elasticity. *Scientific Programming*, 15, 157–172.
Koko J. (2016). Fast MATLAB assembly of fem matrices in 2d and 3d using cell array approach. *International Journal of Modelling and Simulation*, 7.
Kwon, Y.W., Bang, H. (2000). *The Finite Element Method Using MATLAB*. CRC Press, New York.
Matsumoto, J. (2005). A relationship between stabilized FEM and bubble functions element stabilization method with orthogonal basis for incompressible flows. *Journal of Applied Mechanics*, 8, 233–242.
Persson, P.-O., Strang, G. (2004). A simple mesh generator in Matlab. *SIAM Review*, 42, 329–345.
Pierre, R. (1987). *Regularization procedures of mixed finite element approximation of the Stokes problem*. Research-Report RR-0673, INRIA.
Pierre, R. (1989). Regularization procedures of mixed finite element approximation of the Stokes problem. *Numerical Methods for Partial Differential Equations*, 5, 241–258.
Pierre R. (1995). Optimal selection of the bubble function in the stabilization of P1-P1 element for the Stokes problem. *Numerical Methods for Partial Differential Equations*, 32, 1210–1224.
Rahman, T., Valdman, J. (2013). Fast MATLAB assembly of FEM matrices in 2D and 3D: nodal elements. *Applied Mathematics and Computation*, 219, 7151–7158.
Rahman, T., Valdman, J. (2015). Fast MATLAB assembly of FEM matrices in 2D and 3D: edge elements. *Applied Mathematics and Computation*, 267, 252–263.
Schäfer, M., Turek, S. (1996). Benchmark computations of laminar flow around a cylinder. *Notes Numer. Fluid Mech.*, 52:547–566.

J. Koko is an associate professor in applied mathematics at the Computer Science School at the University Clermont-Auvergne. His research interests include numerical optimization with applications to Partial Differential Equations (PDE) and parallel computing, vectorized MATLAB codes for the numerical approximation of PDEs.