Some computational results for $\mathbf{P}_1 - P_1$ and $\mathbf{P}_2 - P_1$ Trace FEM for the surface Stokes problem

Alexander Zhiliakov*

April 19, 2019

Contents

1 Inf-sup stability: generalized pressure Schur complement eigenvalues

1.1 Bilinear forms and matrices

We set $n_{\mathbf{A}}$ to be the number of velocity d.o.f. and $n_{\mathbf{S}}$ to be the number of pressure d.o.f. Vector stiffness, divergence, pressure mass, normal stabilization, and full stabilization matrices resulting from Trace FEM

*Department of Mathematics, University of Houston, Houston, Texas 77204 (alex@math.uh.edu).

discretization of the surface Stokes problem [1] are defined via

$$
\begin{aligned}
\langle \mathbf{A}\,\vec{\mathbf{u}}, \vec{\mathbf{v}} \rangle &\approx \int_\Gamma \left(E_s(\mathbf{u}) : E_s(\mathbf{v}) + \mathbf{u}\cdot\mathbf{v} + \tau\,(\mathbf{u}\cdot\mathbf{n})\,(\mathbf{v}\cdot\mathbf{n}) \right) \mathrm{d}s + \rho_u \int_{\Omega_h^\Gamma} \frac{\partial \mathbf{u}}{\partial \mathbf{n}} \cdot \frac{\partial \mathbf{v}}{\partial \mathbf{n}}\,\mathrm{d}\mathbf{x}, \quad \mathbf{A} \in \mathbb{R}^{n_\mathbf{A} \times n_\mathbf{A}}, \\
\langle \mathbf{B}\,\vec{\mathbf{u}}, \vec{\mathbf{q}} \rangle &\approx -\int_\Gamma q\ \mathrm{div}_\Gamma\,\mathbf{u}\,\mathrm{d}s, \quad \mathbf{B} \in \mathbb{R}^{n_\mathbf{S} \times n_\mathbf{A}}, \\
\langle \mathbf{M}_0\,\vec{\mathbf{p}}, \vec{\mathbf{q}} \rangle &\approx \int_\Gamma p\,q\,\mathrm{d}s, \quad \mathbf{M}_0 \in \mathbb{R}^{n_\mathbf{S} \times n_\mathbf{S}}, \\
\langle \mathbf{C}_n\,\vec{\mathbf{p}}, \vec{\mathbf{q}} \rangle &\approx \rho_p \int_{\Omega_h^\Gamma} \frac{\partial p}{\partial \mathbf{n}} \frac{\partial q}{\partial \mathbf{n}}\,\mathrm{d}\mathbf{x}, \quad \mathbf{C}_n \in \mathbb{R}^{n_\mathbf{S} \times n_\mathbf{S}}, \\
\langle \mathbf{C}_{\text{full}}\,\vec{\mathbf{p}}, \vec{\mathbf{q}} \rangle &\approx \rho_p \int_{\Omega_h^\Gamma} \nabla p \cdot \nabla q\,\mathrm{d}\mathbf{x}, \quad \mathbf{C}_{\text{full}} \in \mathbb{R}^{n_\mathbf{S} \times n_\mathbf{S}},
\end{aligned}
\tag{1}
$$

respectively. We use notations as in [1], in particular, Ω_Γ^h is the domain consisting of tetrahedra cut by Γ. Here $\vec{\mathbf{u}}$ denotes a vector of d.o.f. corresponding to a FE interpolant $\mathbf{u}$ (analogously for $\vec{\mathbf{p}}$ and p). See (16) and (17) for the computational details. Mesh-dependent parameters are set as

$$
\tau = h^{-2}, \quad \rho_u = \rho_p = h, \tag{2}
$$

and h is the typical mesh size for tetrahedra from Ω_h^Γ. Γ is chosen either as the unit sphere or torus, $\Gamma = \Gamma_{\text{sph}}$ or $\Gamma = \Gamma_{\text{tor}}$ (see Figure 1).

We also define matrices

$$
\mathbf{C}_0 \coloneqq \mathbf{0}, \quad \mathbf{M}_n \coloneqq \mathbf{M}_0 + \mathbf{C}_n, \quad \mathbf{M}_{\text{full}} \coloneqq \mathbf{M}_0 + \mathbf{C}_{\text{full}}. \tag{3}
$$

We are interested in (generalized) extreme eigenvalues of the pressure Schur complement matrices

$$
\mathbf{S}_0 \coloneqq \mathbf{B}\,\mathbf{A}^{-1}\,\mathbf{B}^T, \quad \mathbf{S}_n \coloneqq \mathbf{S}_0 + \mathbf{C}_n, \quad \mathbf{S}_{\text{full}} \coloneqq \mathbf{S}_0 + \mathbf{C}_{\text{full}}, \tag{4}
$$

i.e. in solving

$$
\mathbf{S}_\star\,\mathbf{x} = \lambda\,\mathbf{M}_\star\,\mathbf{x}, \tag{5}
$$

where "$\star$" stands for "0," "n," or "full." We denote by $0 = \lambda_1 < \lambda_2 \le \cdots \le \lambda_{n_\mathbf{S}} = O(1)$ the spectrum of (5).

1.2 Solution description

Computing $\mathbf{A}^{-1}$ in (4) becomes troublesome already for $h = 5.21 \times 10^{-2}$ ($n_\mathbf{A} = 32736$ for $\mathbf{u} \in \mathbf{P}_1$ FE space): although $\mathbf{A}$ is sparse, $\mathbf{A}^{-1}$ is dense and consumes 8.5+ GB in double-precision arithmetic. A quick research showed that `Mathematica` has no built-in matrix-free eigenvalue routines. `Intel MKL`'s FEAST algorithm for computing (generalized) eigenvalues in an interval is suitable for matrix-free implementations; however, it requires some expensive operations to be implemented (e.g. matrix-matrix multiplications $\mathbf{Y} \leftarrow \mathbf{S}_\star\,\mathbf{X}$, $\mathbf{Y} \leftarrow \mathbf{M}_\star\,\mathbf{X}$ and approximating the action of inverses in the form $\mathbf{y} \leftarrow (\sigma\,\mathbf{M}_\star - \mathbf{S}_\star)^{-1}\,\mathbf{x}$).

Taking this into account, instead of (5) we consider a perturbed[1] problem

$$
\underbrace{\begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & -\mathbf{C}_\star \end{bmatrix}}_{\mathcal{A}_\star \coloneqq} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \mu \underbrace{\begin{bmatrix} \epsilon\,\mathbf{A} & \\ & \mathbf{M}_\star \end{bmatrix}}_{\mathcal{M}_\star^\epsilon \coloneqq} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \tag{6}
$$

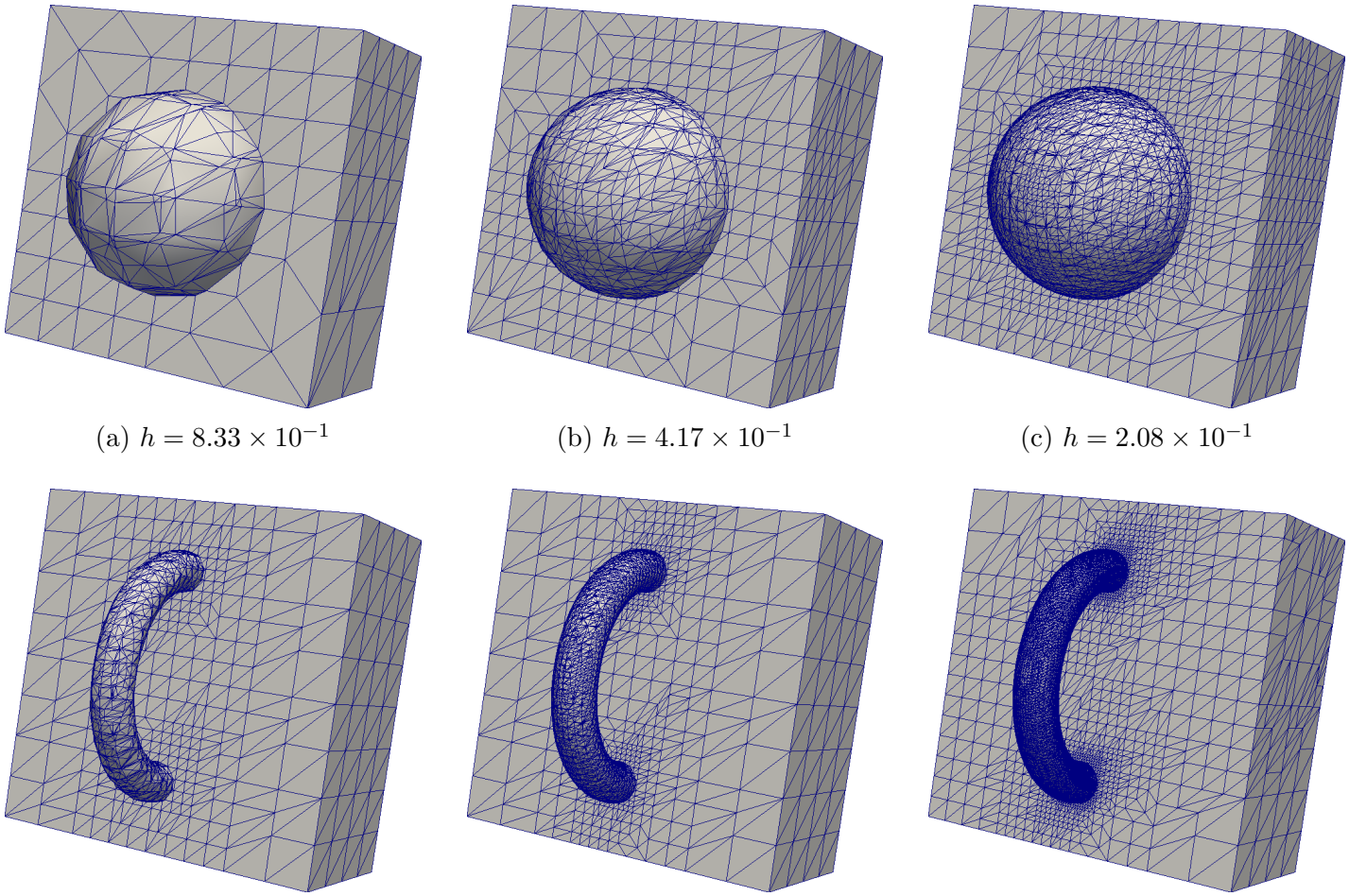
with $0 < \epsilon \ll 1$. For $\mathcal{A}_0$ and $\mathcal{M}_0^\epsilon$ we have

$$
\mu = -\lambda + o(1) \quad \text{or} \quad \epsilon^{-1} + \lambda + o(1), \qquad \epsilon \to 0. \tag{7}
$$

This makes it easy to pick only "correct" eigenvalues. To ease the computation further we replace the $(1,1)$-block of $\mathcal{M}^{\epsilon}_{\star}$ with $\epsilon\,\mathbf{I}$.

To make sure that results are consistent we solve (6) for $\epsilon = 10^{-5}$ and $\epsilon = 10^{-6}$; for the coarse mesh levels we also check that the dense solver for (5) and the iterative one for (6) give solutions that coincide.

1.3 Numerical results: dependency of the spectrum on the mesh size



(a) $h = 8.33 \times 10^{-1}$ (b) $h = 4.17 \times 10^{-1}$ (c) $h = 2.08 \times 10^{-1}$

(d) $h = 2.08 \times 10^{-1}$ (e) $h = 1.04 \times 10^{-1}$ (f) $h = 5.21 \times 10^{-2}$

Figure 1: First three mesh levels for Γ_{sph} (top) and Γ_{tor} (bottom)

[1]The majority of generalized eigenvalue solvers require left-hand-side matrix to be Hermitian and right-hand-side matrix to be Hermitian **positive definite**; that's why we need to introduce $\epsilon > 0$.

Table 1: Spectrum of (5) for $\mathbf{P}_1 - P_1$

(a) $\Gamma = \Gamma_{\text{sph}}$

h	$n_{\mathbf{A}}$	$n_{\mathbf{S}}$	$\mathbf{S}_0$		$\mathbf{S}_n$		$\mathbf{S}_{\text{full}}$	
			λ_2	$\lambda_{n_{\mathbf{S}}}$	λ_2	$\lambda_{n_{\mathbf{S}}}$	λ_2	$\lambda_{n_{\mathbf{S}}}$
8.33×10^{-1}	153	51	1.32×10^{-2}	1.42	7.48×10^{-1}	1.13	9.58×10^{-1}	1.06
4.17×10^{-1}	570	190	5.12×10^{-3}	1.04	5.77×10^{-1}	1.	8.54×10^{-1}	1.
2.08×10^{-1}	1992	664	4.4×10^{-3}	7.93×10^{-1}	3.87×10^{-1}	1.	6.71×10^{-1}	1.
1.04×10^{-1}	8292	2764	2.01×10^{-3}	7.79×10^{-1}	2.19×10^{-1}	1.	5.82×10^{-1}	1.
5.21×10^{-2}	32736	10912	6.04×10^{-5}	9.81×10^{-1}	1.17×10^{-1}	1.	5.37×10^{-1}	1.
2.6×10^{-2}	131592	43864	3.53×10^{-5}	8.67×10^{-1}	5.72×10^{-2}	1.	5.16×10^{-1}	1.
1.3×10^{-2}	525864	175288	2.16×10^{-6}	7.34×10^{-1}	2.84×10^{-2}	1.	5.04×10^{-1}	1.

(b) $\Gamma = \Gamma_{\text{tor}}$

h	$n_{\mathbf{A}}$	$n_{\mathbf{S}}$	$\mathbf{S}_0$		$\mathbf{S}_n$		$\mathbf{S}_{\text{full}}$	
			λ_2	$\lambda_{n_{\mathbf{S}}}$	λ_2	$\lambda_{n_{\mathbf{S}}}$	λ_2	$\lambda_{n_{\mathbf{S}}}$
2.08×10^{-1}	972	324	5.04×10^{-2}	4.93	2.84×10^{-1}	1.35	3.64×10^{-1}	1.19
1.04×10^{-1}	4740	1580	2.99×10^{-3}	3.83	1.58×10^{-1}	1.02	3.35×10^{-1}	1.01
5.21×10^{-2}	19704	6568	1.11×10^{-3}	5.45	7.73×10^{-2}	1.01	3.25×10^{-1}	1.
2.6×10^{-2}	80808	26936	1.2×10^{-4}	5.42	3.07×10^{-2}	1.01	3.21×10^{-1}	1.
1.3×10^{-2}	327036	109012	1.77×10^{-5}	5.23	1.18×10^{-2}	1.01	3.16×10^{-1}	1.

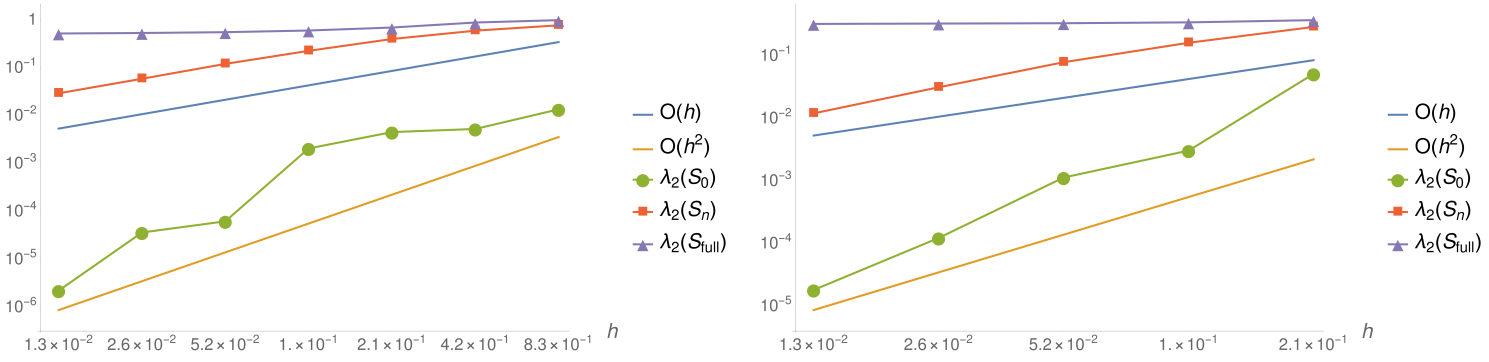


Figure 2: Log-log plot of λ_2 for Tables 1a (left) and 1b (right)

Table 2: Spectrum of (5) for $\mathbf{P}_2 - P_1$

(a) $\Gamma = \Gamma_{\text{sph}}$

h	$n_{\mathbf{A}}$	$n_{\mathbf{S}}$	$\mathbf{S}_0$		$\mathbf{S}_n$		$\mathbf{S}_{\text{full}}$	
			λ_2	$\lambda_{n_{\mathbf{S}}}$	λ_2	$\lambda_{n_{\mathbf{S}}}$	λ_2	$\lambda_{n_{\mathbf{S}}}$
8.33×10^{-1}	789	51	3.22×10^{-1}	1.73	8.27×10^{-1}	1.17	9.68×10^{-1}	1.07
4.17×10^{-1}	3240	190	9.17×10^{-2}	1.08	6.45×10^{-1}	1.	8.56×10^{-1}	1.
2.08×10^{-1}	11718	664	1.78×10^{-1}	8.31×10^{-1}	5.49×10^{-1}	1.	6.75×10^{-1}	1.
1.04×10^{-1}	48762	2764	1.04×10^{-1}	8.35×10^{-1}	5.14×10^{-1}	1.	5.82×10^{-1}	1.
5.21×10^{-2}	193014	10912	2.99×10^{-3}	9.89×10^{-1}	5.02×10^{-1}	1.	5.34×10^{-1}	1.
2.6×10^{-2}	775998	43864	1.17×10^{-3}	7.9×10^{-1}	4.96×10^{-1}	1.	5.17×10^{-1}	1.

(b) $\Gamma = \Gamma_{\text{tor}}$

h	$n_{\mathbf{A}}$	$n_{\mathbf{S}}$	$\mathbf{S}_0$		$\mathbf{S}_n$		$\mathbf{S}_{\text{full}}$	
			λ_2	$\lambda_{n_{\mathbf{S}}}$	λ_2	$\lambda_{n_{\mathbf{S}}}$	λ_2	$\lambda_{n_{\mathbf{S}}}$
2.08×10^{-1}	5184	324	9.92×10^{-2}	3.89	1.33×10^{-1}	1.37	1.75×10^{-1}	1.19
1.04×10^{-1}	27906	1580	1.46×10^{-2}	4.35	2.84×10^{-1}	1.04	2.99×10^{-1}	1.02
5.21×10^{-2}	116568	6568	6.08×10^{-3}	4.85	3.19×10^{-1}	1.01	3.24×10^{-1}	1.01
2.6×10^{-2}	477660	26936	1.36×10^{-3}	4.92	3.14×10^{-1}	1.01	3.16×10^{-1}	1.

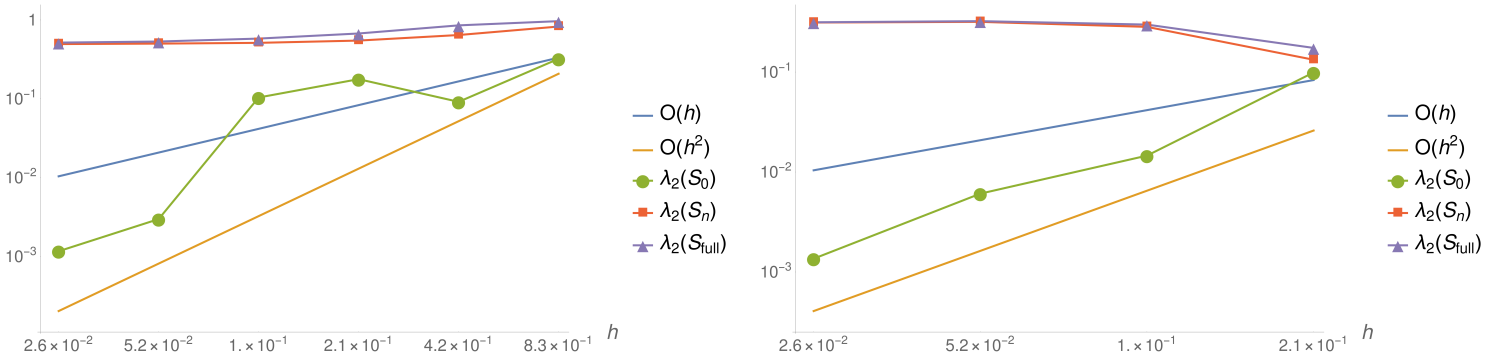


Figure 3: Log-log plot of λ_2 for Tables 2a (left) and 2b (right)

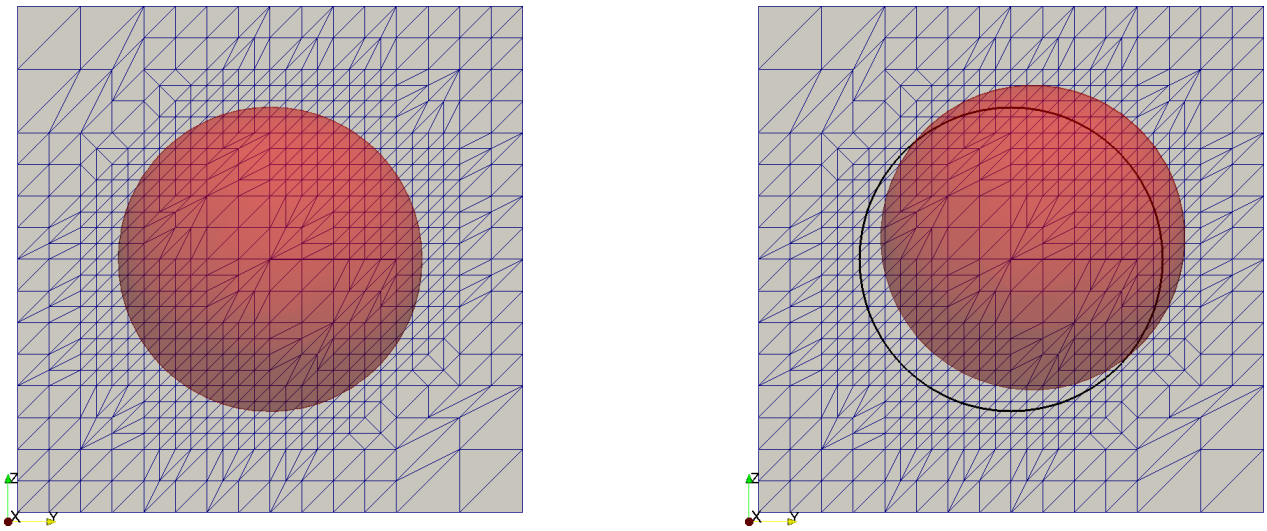
1.4 Numerical results: sensitivity of the spectrum to levelset shifts

In this section we investigate the sensitivity of the spectrum to levelset shifts

$$\Gamma \mapsto \Gamma + \alpha\,\mathbf{s}, \tag{8}$$

for some $\alpha \in \mathbb{R}$ and $\mathbf{s} \in \mathbb{R}^3$, $\|\mathbf{s}\| = 1$.

We construct the bulk mesh Ω_h^Γ and then perform the assembly of matrices (1) using the shifted levelset (8). That is, the refinement of Ω_h^Γ is performed using Γ, not $\Gamma + \alpha\,\mathbf{s}$, and $\Omega_h^{\Gamma+\alpha\,\mathbf{s}}$ is never constructed. We choose $\alpha \in [0, h]$ to guarantee the appearance of "small cuts" in Ω_h^Γ.



(a) Γ_{sph} (b) $\Gamma_{\text{sph}} + \alpha\,\mathbf{s}$

Figure 4: The unit sphere (left) and the shifted unit sphere (right). Here $\mathbf{s} = (0, 1, 1)^T/\sqrt{2}$, $\alpha = 0.2$, and $h = 2.08 \times 10^{-1}$. The bulk mesh Ω_Γ^h is computed for Γ_{sph} and then used for $\Gamma_{\text{sph}} + \alpha\,\mathbf{s}$

Table 3: Spectrum of (5) for perturbed levelset $\Gamma_{\text{sph}} + \alpha\,\mathbf{s}$. Here $\mathbf{s} = (1,1,1)^T/\sqrt{3}$, $h = 1.04 \times 10^{-1}$

(a) $\mathbf{P}_1 - P_1$

Surface	$\mathbf{S}_0$		$\mathbf{S}_n$		$\mathbf{S}_{\text{full}}$	
	λ_2	$\lambda_{n_\mathbf{S}}$	λ_2	$\lambda_{n_\mathbf{S}}$	λ_2	$\lambda_{n_\mathbf{S}}$
Γ_{sph}	2.006×10^{-3}	7.79×10^{-1}	2.19×10^{-1}	1.	5.818×10^{-1}	1.
$\Gamma_{\text{sph}} + 0.1\,h\,\mathbf{s}$	4.832×10^{-4}	8.01×10^{-1}	2.195×10^{-1}	1.	5.818×10^{-1}	1.
$\Gamma_{\text{sph}} + 0.3\,h\,\mathbf{s}$	7.278×10^{-4}	8.17×10^{-1}	2.203×10^{-1}	1.	5.818×10^{-1}	1.
$\Gamma_{\text{sph}} + 0.5\,h\,\mathbf{s}$	3.121×10^{-4}	8.67×10^{-1}	2.221×10^{-1}	1.	5.82×10^{-1}	1.
$\Gamma_{\text{sph}} + 0.7\,h\,\mathbf{s}$	1.438×10^{-3}	1.51	2.254×10^{-1}	1.	5.82×10^{-1}	1.
$\Gamma_{\text{sph}} + \quad h\,\mathbf{s}$	1.79×10^{-3}	2.07	2.332×10^{-1}	1.	5.827×10^{-1}	1.

(b) $\mathbf{P}_2 - P_1$

Surface	$\mathbf{S}_0$		$\mathbf{S}_n$		$\mathbf{S}_{\text{full}}$	
	λ_2	$\lambda_{n_\mathbf{S}}$	λ_2	$\lambda_{n_\mathbf{S}}$	λ_2	$\lambda_{n_\mathbf{S}}$
Γ_{sph}	1.041×10^{-1}	8.35×10^{-1}	5.138×10^{-1}	1.	5.841×10^{-1}	1.
$\Gamma_{\text{sph}} + 0.1\,h\,\mathbf{s}$	1.705×10^{-3}	8.58×10^{-1}	5.138×10^{-1}	1.	5.841×10^{-1}	1.
$\Gamma_{\text{sph}} + 0.3\,h\,\mathbf{s}$	2.293×10^{-3}	8.81×10^{-1}	5.137×10^{-1}	1.	5.841×10^{-1}	1.
$\Gamma_{\text{sph}} + 0.5\,h\,\mathbf{s}$	5.63×10^{-3}	9.35×10^{-1}	5.138×10^{-1}	1.	5.844×10^{-1}	1.
$\Gamma_{\text{sph}} + 0.7\,h\,\mathbf{s}$	8.188×10^{-3}	1.77	5.138×10^{-1}	1.	5.843×10^{-1}	1.
$\Gamma_{\text{sph}} + \quad h\,\mathbf{s}$	1.93×10^{-2}	2.21	5.142×10^{-1}	1.	5.852×10^{-1}	1.

2 Convergence results

We solve model problem from [1, p. 20]. We set

$$\begin{aligned} \tilde{\mathbf{u}}(x,y,z) &\coloneqq (-z^2, y, x)^T, \\ \tilde{p}(x,y,z) &\coloneqq x\,y^2 + z, \\ \phi(\mathbf{x}) &\coloneqq \|\mathbf{x}\|^2 - 1. \end{aligned} \tag{9}$$

The exact solution on the unit sphere is chosen as

$$\mathbf{u}(\mathbf{x}) \coloneqq \mathbf{P}\,\tilde{\mathbf{u}}\Big(\frac{\mathbf{x}}{\|\mathbf{x}\|}\Big), \quad p(\mathbf{x}) \coloneqq \tilde{p}\Big(\frac{\mathbf{x}}{\|\mathbf{x}\|}\Big). \tag{10}$$

This way we "extend" (9) radially. It is useful since we use *the approximation* of Γ_{sph}.

2.1 $\mathbf{P}_1 - P_1$ Trace FEM

Here we compare approaches (16) and (17). We use one virtual refinement for surface integrals, $m = 2$, so we have $\Gamma^{2\to1}_{h/2} = \Gamma^1_{h/2}$ (see (13) and (15)). We use the full stabilization matrix $\mathbf{C}_{\text{full}}$. Note that $\phi \in P_2$ in (9), and hence $\Gamma^2_h = \Gamma$. Thus (16) boils down to

$$\begin{aligned} \langle \mathbf{A}\,\vec{\mathbf{u}}, \vec{\mathbf{v}}\rangle &= \int_{\Gamma^1_{h/2}}^5 \big(E_{s,\Gamma}(\mathbf{u}) : E_{s,\Gamma}(\mathbf{v}) + \mathbf{u}\cdot\mathbf{v} + \tau\,(\mathbf{u}\cdot\mathbf{n}_\Gamma)\,(\mathbf{v}\cdot\mathbf{n}_\Gamma)\big)\,\mathrm{d}s \\ &\quad + \rho_u \int_{\Omega^\Gamma_h}^5 \frac{\partial \mathbf{u}}{\partial \mathbf{n}_\Gamma}\cdot\frac{\partial \mathbf{v}}{\partial \mathbf{n}_\Gamma}\,\mathrm{d}\mathbf{x}, \quad \mathbf{A}\in\mathbb{R}^{n_\mathbf{A}\times n_\mathbf{A}}, \\ \langle \mathbf{B}\,\vec{\mathbf{u}}, \vec{\mathbf{q}}\rangle &= -\int_{\Gamma^1_{h/2}}^5 q\ \mathrm{div}_\Gamma\,\mathbf{u}\,\mathrm{d}s, \quad \mathbf{B}\in\mathbb{R}^{n_\mathbf{S}\times n_\mathbf{A}}, \\ \langle \mathbf{M}_0\,\vec{\mathbf{p}}, \vec{\mathbf{q}}\rangle &= \int_{\Gamma^1_{h/2}}^5 p\,q\,\mathrm{d}s, \quad \mathbf{M}_0\in\mathbb{R}^{n_\mathbf{S}\times n_\mathbf{S}}, \\ \langle \mathbf{C}_{\text{full}}\,\vec{\mathbf{p}}, \vec{\mathbf{q}}\rangle &= \rho_p \int_{\Omega^\Gamma_h}^5 \nabla p\cdot\nabla q\,\mathrm{d}\mathbf{x}, \quad \mathbf{C}_{\text{full}}\in\mathbb{R}^{n_\mathbf{S}\times n_\mathbf{S}}, \end{aligned} \tag{11}$$

and e.g. the integrand that involves $E_{s,\Gamma} \equiv E_s$ is exact. Similarly, approach (17) boils down to

$$\begin{aligned} \langle \mathbf{A}\,\vec{\mathbf{u}}, \vec{\mathbf{v}}\rangle &= \int_{\Gamma^1_{h/2}}^5 \big(E_{s,\Gamma^1_{h/2}}(\mathbf{u}) : E_{s,\Gamma^1_{h/2}}(\mathbf{v}) + \mathbf{u}\cdot\mathbf{v} + \tau\,(\mathbf{u}\cdot\mathbf{n}_\Gamma)\,(\mathbf{v}\cdot\mathbf{n}_\Gamma)\big)\,\mathrm{d}s \\ &\quad + \rho_u \int_{\Omega^\Gamma_h}^5 \frac{\partial \mathbf{u}}{\partial \mathbf{n}_\Gamma}\cdot\frac{\partial \mathbf{v}}{\partial \mathbf{n}_\Gamma}\,\mathrm{d}\mathbf{x}, \quad \mathbf{A}\in\mathbb{R}^{n_\mathbf{A}\times n_\mathbf{A}}, \\ \langle \mathbf{B}\,\vec{\mathbf{u}}, \vec{\mathbf{q}}\rangle &= -\int_{\Gamma^1_{h/2}}^5 q\ \mathrm{div}_{\Gamma^1_{h/2}}\,\mathbf{u}\,\mathrm{d}s, \quad \mathbf{B}\in\mathbb{R}^{n_\mathbf{S}\times n_\mathbf{A}}, \\ \langle \mathbf{M}_0\,\vec{\mathbf{p}}, \vec{\mathbf{q}}\rangle &= \int_{\Gamma^1_{h/2}}^5 p\,q\,\mathrm{d}s, \quad \mathbf{M}_0\in\mathbb{R}^{n_\mathbf{S}\times n_\mathbf{S}}, \\ \langle \mathbf{C}_{\text{full}}\,\vec{\mathbf{p}}, \vec{\mathbf{q}}\rangle &= \rho_p \int_{\Omega^\Gamma_h}^5 \nabla p\cdot\nabla q\,\mathrm{d}\mathbf{x}, \quad \mathbf{C}_{\text{full}}\in\mathbb{R}^{n_\mathbf{S}\times n_\mathbf{S}}. \end{aligned} \tag{12}$$

Table 4: Convergence results. Matrices are assembled as in (11) (top table) and (12) (bottom table)

m	h	$\|\mathbf{u}-\mathbf{u}_h\|_{\mathbb{H}^1(\Gamma)}$	Order	$\|\mathbf{u}-\mathbf{u}_h\|_{\mathbb{L}^2(\Gamma)}$	Order	$\|p-p_h\|_{\mathbb{L}^2(\Gamma)}$	Order
2	8.33×10^{-1}	3.4		2.2		1.1	
	4.17×10^{-1}	1.8	9.39×10^{-1}	1.1	9.57×10^{-1}	9.3×10^{-1}	2.56×10^{-1}
	2.08×10^{-1}	7.6×10^{-1}	1.23	3.6×10^{-1}	1.65	5.1×10^{-1}	8.75×10^{-1}
	1.04×10^{-1}	3.1×10^{-1}	1.3	$1.\times10^{-1}$	1.85	1.8×10^{-1}	1.46
	5.21×10^{-2}	1.3×10^{-1}	1.21	2.6×10^{-2}	1.95	5.3×10^{-2}	1.79
	2.6×10^{-2}	6.4×10^{-2}	1.05	6.5×10^{-3}	1.98	1.5×10^{-2}	1.84
	1.3×10^{-2}	3.2×10^{-2}	1.01	1.7×10^{-3}	1.97	6.6×10^{-3}	1.17
	6.51×10^{-3}	1.6×10^{-2}	9.93×10^{-1}	5.1×10^{-4}	1.72	5.5×10^{-3}	2.59×10^{-1}
4	6.51×10^{-3}	1.7×10^{-2}	9.2×10^{-1}	4.1×10^{-4}	2.02	1.6×10^{-3}	2.06

h	$\|\mathbf{u}-\mathbf{u}_h\|_{\mathbb{H}^1(\Gamma)}$	Order	$\|\mathbf{u}-\mathbf{u}_h\|_{\mathbb{L}^2(\Gamma)}$	Order	$\|p-p_h\|_{\mathbb{L}^2(\Gamma)}$	Order
8.33×10^{-1}	3.4		2.2		1.1	
4.17×10^{-1}	1.8	9.28×10^{-1}	1.1	9.3×10^{-1}	9.3×10^{-1}	2.58×10^{-1}
2.08×10^{-1}	7.6×10^{-1}	1.23	3.6×10^{-1}	1.65	5.1×10^{-1}	8.62×10^{-1}
1.04×10^{-1}	3.1×10^{-1}	1.29	$1.\times10^{-1}$	1.85	1.9×10^{-1}	1.44
5.21×10^{-2}	1.3×10^{-1}	1.22	2.6×10^{-2}	1.94	5.5×10^{-2}	1.77
2.6×10^{-2}	6.4×10^{-2}	1.07	6.7×10^{-3}	1.97	1.5×10^{-2}	1.89
1.3×10^{-2}	3.1×10^{-2}	1.02	1.7×10^{-3}	1.98	$4.\times10^{-3}$	1.89
6.51×10^{-3}	1.5×10^{-2}	1.02	4.3×10^{-4}	1.99	1.2×10^{-3}	1.77

Table 5: Solver statistics for (11) (left table) and (12) (right table)

h	Outer iterations	Residual norm
8.33×10^{-1}	14	$1.\times10^{-8}$
4.17×10^{-1}	20	9.2×10^{-9}
2.08×10^{-1}	26	8.8×10^{-9}
1.04×10^{-1}	29	5.4×10^{-9}
5.21×10^{-2}	29	7.1×10^{-9}
2.6×10^{-2}	29	4.5×10^{-9}
1.3×10^{-2}	27	9.5×10^{-9}
6.51×10^{-3}	30	6.5×10^{-9}

h	Outer iterations	Residual norm
8.33×10^{-1}	15	1.7×10^{-9}
4.17×10^{-1}	20	7.7×10^{-9}
2.08×10^{-1}	26	$7.\times10^{-9}$
1.04×10^{-1}	29	4.3×10^{-9}
5.21×10^{-2}	29	5.2×10^{-9}
2.6×10^{-2}	27	8.5×10^{-9}
1.3×10^{-2}	27	3.6×10^{-9}
6.51×10^{-3}	29	7.4×10^{-9}

For statistics: using 64 CPUs, computation of the meshlevel 6 ($h = 2.6\times10^{-2}$) takes ∼14 minutes, meshlevel 7 takes ∼75 minutes, and meshlevel 8 takes ∼7.3 hours.

The errors in Table 4 are computed as explained in section 3.6.

2.2 $\mathbf{P}_2 - P_1$ Trace FEM

We use the normal stabilization matrix $\mathbf{C}_n$. We stick to approach (17). We choose $m = O(h^{-1/2})$ so that the geometric error is $O(h^3)$. Thus $m = 2, 2, 4, 4, 6, 8, 10, 14$ for meshlevel 1, 2 and so forth, respectively.

Table 6: Convergence results. $\tau = h^{-2}$, $\rho_u = \rho_p = h$

h	$\|\mathbf{u} - \mathbf{u}_h\|_{\mathbb{H}^1(\Gamma)}$	Order	$\|\mathbf{u} - \mathbf{u}_h\|_{\mathbb{L}^2(\Gamma)}$	Order	$\|p - p_h\|_{\mathbb{L}^2(\Gamma)}$	Order
8.33×10^{-1}	2.8		1.7		2.1	
4.17×10^{-1}	1.9	5.74×10^{-1}	$9. \times 10^{-1}$	9.15×10^{-1}	1.7	2.57×10^{-1}
2.08×10^{-1}	7.2×10^{-1}	1.37	3.4×10^{-1}	1.39	7.1×10^{-1}	1.31
1.04×10^{-1}	2.2×10^{-1}	1.7	$1. \times 10^{-1}$	1.78	2.1×10^{-1}	1.76
5.21×10^{-2}	6.2×10^{-2}	1.85	2.6×10^{-2}	1.92	5.1×10^{-2}	2.04
2.6×10^{-2}	1.8×10^{-2}	1.81	6.5×10^{-3}	2.01	1.3×10^{-2}	1.91

h	$\|\mathbf{u}_h \cdot \mathbf{n}\|_{\mathbb{L}^2(\Gamma)}$	Order	Outer iterations	Residual norm
8.33×10^{-1}	1.7		24	9.9×10^{-9}
4.17×10^{-1}	8.9×10^{-1}	8.87×10^{-1}	31	4.6×10^{-9}
2.08×10^{-1}	3.4×10^{-1}	1.38	31	4.5×10^{-9}
1.04×10^{-1}	$1. \times 10^{-1}$	1.78	29	3.5×10^{-9}
5.21×10^{-2}	2.6×10^{-2}	1.95	27	6.9×10^{-9}
2.6×10^{-2}	6.5×10^{-3}	1.99	28	7.3×10^{-9}

Table 7: Convergence results. $\tau = h^{-3}$, $\rho_u = \rho_p = h$

h	$\|\mathbf{u} - \mathbf{u}_h\|_{\mathbb{H}^1(\Gamma)}$	Order	$\|\mathbf{u} - \mathbf{u}_h\|_{\mathbb{L}^2(\Gamma)}$	Order	$\|p - p_h\|_{\mathbb{L}^2(\Gamma)}$	Order
8.33×10^{-1}	2.7		1.6		2.	
4.17×10^{-1}	1.2	1.2	5.3×10^{-1}	1.61	1.	9.82×10^{-1}
2.08×10^{-1}	2.4×10^{-1}	2.26	8.5×10^{-2}	2.66	1.9×10^{-1}	2.44
1.04×10^{-1}	7.5×10^{-2}	1.68	1.2×10^{-2}	2.87	2.8×10^{-2}	2.74
5.21×10^{-2}	$3. \times 10^{-2}$	1.32	5.1×10^{-3}	1.19	4.6×10^{-3}	2.62
2.6×10^{-2}	1.1×10^{-2}	1.43	$3. \times 10^{-4}$	4.09	1.4×10^{-3}	1.69
1.3×10^{-2}	4.7×10^{-3}	1.24	6.6×10^{-5}	2.18	6.2×10^{-4}	1.18

h	$\|\mathbf{u}_h \cdot \mathbf{n}\|_{\mathbb{L}^2(\Gamma)}$	Order	Outer iterations	Residual norm
8.33×10^{-1}	1.6		25	4.4×10^{-9}
4.17×10^{-1}	5.2×10^{-1}	1.61	32	3.6×10^{-9}
2.08×10^{-1}	8.4×10^{-2}	2.63	31	5.9×10^{-9}
1.04×10^{-1}	1.1×10^{-2}	2.91	28	9.6×10^{-9}
5.21×10^{-2}	1.4×10^{-3}	3.01	27	7.4×10^{-9}
2.6×10^{-2}	1.7×10^{-4}	3.01	27	3.3×10^{-9}
1.3×10^{-2}	2.1×10^{-5}	3.	33	8.8×10^{-9}

For statistics: using 64 CPUs, computation of the meshlevel 3 ($h = 2.08 \times 10^{-1}$) takes ∼1 minute, meshlevel 4 takes ∼7 minutes, meshlevel 5 takes ∼50 minutes, meshlevel 6 takes 4.8 hours, and meshlevel 7 takes ∼21.3 hours.

3 Notes on DROPS implementation

3.1 Notations

We denote by $P_h^n \subset \bar{P}_h^n$ spaces of continuous and discontinuous nodal P_n interpolants defined on Ω_Γ^h, respectively. For a function f, $I_h^n(f) \in P_h^n$ is the corresponding interpolant; we will use the notation f_h^n to emphasize that $f_h^n \in P_h^n$ and f_h^n approximates f in some sense, but $I_h^n(f) \neq f_h^n$.

We set

$$\Gamma_h^n \coloneqq \{\mathbf{x} \in \mathbb{R}^3 : \big(I_h^n(\phi)\big)(\mathbf{x}) = 0\}, \tag{13}$$

$$\mathbf{n}_{\Gamma_h^n} = \frac{\nabla I_h^n(\phi)}{\|\nabla I_h^n(\phi)\|} \notin \bar{P}_h^m \text{ for any } m \text{ if } n > 1. \tag{14}$$

Note that Γ_h^n is a continuous piecewise P_n surface in Ω_Γ^h, and $\Gamma_h^n \neq I_h^n(\Gamma)$. The unit normal $\mathbf{n}_{\Gamma_h^n}$ is not a rational function; it is continuous in $T \in \Omega_\Gamma^h$ and discontinuous on faces. We also define

$$\Gamma_{h/m}^{2\to1} \coloneqq \{\mathbf{x} \in \mathbb{R}^3 : \Big(I_{h/m}^1\big(I_h^2(\phi)\big)\Big)(\mathbf{x}) = 0\}. \tag{15}$$

Note that $I_{h/2}^1\big(I_h^2(\phi)\big) = I_{h/2}^1(\phi)$ (since in order to build both $I_{h/2}^1$ and I_h^2 the same values of ϕ are used), and $I_{h/m}^1\big(I_h^2(\phi)\big) \neq I_{h/m}^1(\phi)$ for $m > 2$. Thus we have $\Gamma_{h/2}^{2\to1} = \Gamma_{h/2}^1$, and $\Gamma_{h/m}^{2\to1} \neq \Gamma_{h/m}^1$ for $m > 2$.

3.2 Approximation of integrands involving $\mathbf{n}_\Gamma$

We start with description of the continuous levelset ϕ of $\Gamma = \{\mathbf{x} \in \mathbb{R}^3 : \phi(\mathbf{x}) = 0\}$. It is stored in `levelset_fun` variable. For example, for the unit sphere we have:

```
// surfnavierstokes_funcs.h
DROPS::Point3DCL sphere_2_shift(0.);
double sphere_2 (const DROPS::Point3DCL& p, double) {
  return pow(p[0] - sphere_2_shift[0], 2.) +
         pow(p[1] - sphere_2_shift[1], 2.) +
         pow(p[2] - sphere_2_shift[2], 2.) - 1.;
}

// surfnavierstokes.cpp
instat_scalar_fun_ptr levelset_fun;
// ...
levelset_fun = &sphere_2;
```

Continuous piecewise P_2 interpolant $I_h^2(\phi)$ of ϕ is built on Ω_Γ^h via iterating over vertices and edges of Ω_Γ^h. It is stored in `lset` object:

```
// levelset.cpp
void LevelsetP2ContCL::Init( instat_scalar_fun_ptr phi0, double t) {
  const Uint lvl= Phi.GetLevel(),
  idx= Phi.RowIdx->GetIdx();
```

```
  for (auto it = MG_.GetTriangVertexBegin(lvl), end = MG_.GetTriangVertexEnd(lvl); it
      != end; ++it) {
    if (it->Unknowns.Exist(idx))
      Phi.Data[it->Unknowns(idx)]= phi0( it->GetCoord(), t);
  }
  for (auto it = MG_.GetTriangEdgeBegin(lvl), end = MG_.GetTriangEdgeEnd(lvl); it !=
      end; ++it) {
    if (it->Unknowns.Exist(idx))
      Phi.Data[it->Unknowns(idx)]= phi0( GetBaryCenter( *it), t);
  }
}

// surfnavierstokes.cpp
DROPS::LevelsetP2CL& lset(*DROPS::LevelsetP2CL::Create(mg, lsbnd, sf));
// ...
lset.Init(levelset_fun);
```

In order to assemble matrices in (1) for e.g. $\mathbf{P}_1 - P_1$ elements, one calls

```
SetupNavierStokesIF_P1P1(mg, &A, /* ... */ lset.Phi, /* ... */);
```

(**Interestingly enough**, this function does not get `lset` object that represents the interpolant; it gets only `lset.Phi`, which is the object of type `VecDescCL`. `lset.Phi` is essentially just a vector of values of ϕ at interpolation points (i.e. vertices and edges' centroids of Ω_Γ^h). That is, the assembling function above has no idea what `lset.Phi` actually represents: one may interpret it as an element of P_h^2 or e.g. $P_{h/2}^1$. Who knows?..)

No interpolation is built explicitly for $\mathbf{n}_{\Gamma_h^2}$ in (14); it is implicitly represented via `qnormal` data field:

```
// ifacetransp.cpp
class LocalStokesCL {
  // ...
  GridFunctionCL<Point3DCL> qnormal;
  // ...
}
```

`qnormal` object is essentially a set of values of type `Point3DCL` which are obtained by mapping a (vector valued) function to suitable quadrature nodes. This is how it is constructed:

```
// ifacetransp.cpp
void LocalStokesCL::Get_Normals(const LocalP2CL<>& ls, LocalP1CL<Point3DCL>& Normals) {
  for(int i=0; i<10 ; ++i)
    Normals+=ls[i]*P2Grad[i];
}
// ...
void LocalStokesCL::calcIntegrands(const SMatrixCL<3,3>& T, const LocalP2CL<>& ls,
    const TetraCL& tet) {
  // ...
  LocalP1CL<Point3DCL> Normals;
  Get_Normals(ls, Normals);
  resize_and_evaluate_on_vertexes (Normals, q2Ddomain, qnormal);
  for(Uint i=0; i<qnormal.size(); ++i)
    qnormal[i]= qnormal[i]/qnormal[i].norm();
  // ...
}
```

First $\nabla I_h^2(\phi|_T)$ is built (locally for a tetrahedron $T \in \Omega_\Gamma^h$ represented by `tet`) and saved to `Normals` object. `ls[i]` gives the value of $\phi|_T$ at ith node (vertices and edges' centroids — there are 10 of them for tetrahedra), and `P2Grad[i]` represents the gradient of quadratic basis function which itself is linear. (Actually, it is sufficient to have $4 < 10$ linear functions to represent $\nabla I_h^2(\phi|_T)$, but this is how it is implemented here.)

Finally, `qnormal` object is built via evaluating `Normals` at quadrature nodes and normalization.

Objects `qnormal` for surface integrals and `q3Dnormal` for volume integrals are used in approximation of $\mathbf{P} = \mathbf{I} - \mathbf{n}\,\mathbf{n}^T$, normal derivatives, and taking-normal-components in (1). `q3Dnormal` is constructed as `qnormal` but for quadrature points of tetrahedrons, not triangles.

For one, $\mathbf{P}\,\nabla f_h^2$, $f_h^2 \coloneqq P_2$ basis function on Ω_Γ^h, is approximated via `qsurfP2grad` object:

```
// ifacetransp.cpp
void LocalStokesCL::calcIntegrands(/* ... */) {
    // ...
    for(int j=0; j<10 ;++j) {
    resize_and_evaluate_on_vertexes(P2Grad[j], q2Ddomain, qsurfP2grad[j]);
    qsurfP2grad[j]-= dot(qsurfP2grad[j], qnormal)*qnormal;
  }
  // ...
}
```

The term $\int_{\Omega_h^\Gamma} \frac{\partial p}{\partial \mathbf{n}} \frac{\partial q}{\partial \mathbf{n}}\,\mathrm{d}\mathbf{x}$ in (1) is computed as

```
// ifacetransp.cpp
void LocalStokesCL::setupA_P1_stab(double A_P1_stab[4][4], double absdet) {
  for (int i=0; i<4; ++i)
    for (int j=0; j<4; ++j)
      A_P1_stab[i][j] = quad(dot(q3Dnormal, q3DP1Grad[i])*dot(q3Dnormal, q3DP1Grad[j]),
          absdet, q3Ddomain, AllTetraC);
}
```

3.3 Quadrature rules for $\int_\Gamma$ and $\int_{\Omega_\Gamma^h}$

All the 3D integrals in (1) are computed via iteration over $T \in \Omega_\Gamma^h$ without any virtual refinements. `q3Ddomain` object represents the set of quadrature nodes and weights:

```
// ifacetransp.cpp
void LocalStokesCL::calc3DIntegrands(/* ... */) {
  make_SimpleQuadDomain<Quad5DataCL> (q3Ddomain, AllTetraC);
  // ...
}
```

It is used e.g. in `setupA_P1_stab` above. 15 nodes and weights are used, and the quadrature is exact for functions in $\bar{P}_h^5$.

All the surface integrals are also computed via iteration over $T \in \Omega_\Gamma^h$, but using $\Gamma_{h/2}^1$. One extra "virtual" refinement is achieved via setting

```
// ifacetransp.cpp
LocalStokesCL(bool fullGradient)
  : lat(PrincipalLatticeCL::instance(2))
  , /* ... */ { /* ... */ }
```

`PrincipalLatticeCL::instance(2)` means that each edge of the tetrahedron $T \in \Omega_\Gamma^h$ is split into 2 edges, and T is split into 8 smaller tetrahedrons. Changing 2 to 4 will give us $\Gamma_{h/4}^{2\to1}$ from (15) and so forth. `q2Ddomain` object represents the set of quadrature nodes and weights:

```
// ifacetransp.cpp
void LocalStokesCL::calcIntegrands(/* ... */) {
  // ...
  evaluate_on_vertexes( ls, lat, Addr( ls_loc));
  spatch.make_patch<MergeCutPolicyCL>( lat, ls_loc);
  make_CompositeQuad5Domain2D ( q2Ddomain, spatch, tet);
```

```
  // ...
}
```

Each linear subsurface in $T \in \Omega_\Gamma^h$ has 7 quadrature nodes and weights, and the quadrature rule is again exact for functions in $\bar{P}_h^5$.

`spatch` represents a set of triangles that form $\Gamma_{h/2}^1$ inside T. That is, in order to approximate zeros of ϕ, $I_{h/2}^1\big(I_h^2(\phi)\big) = I_{h/2}^1(\phi)$ is used:

```
// subtriangulation.h
// ...
const double edge_bary1_cut= ls0/(ls0 - ls1); // the root of the level set function on
    the edge
// ...
```

Here $l(x) := \texttt{ls0}\,(1-x) + \texttt{ls1}\,x$ is a linear function defined on the master edge $[0,1]$. Indeed, its root is $x = \texttt{ls0/(ls0 - ls1)}$.

3.4 Summary on the matrix assembly

The matrices in (1) are assembled as

$$
\begin{aligned}
\langle \mathbf{A}\,\vec{\mathbf{u}}, \vec{\mathbf{v}} \rangle &= \int_{\Gamma_{h/m}^{2\to 1}}^{5} \left(E_{s,\,\Gamma_h^2}(\mathbf{u}) : E_{s,\,\Gamma_h^2}(\mathbf{v}) + \mathbf{u}\cdot\mathbf{v} + \tau\,(\mathbf{u}\cdot\mathbf{n}_{\Gamma_h^2})\,(\mathbf{v}\cdot\mathbf{n}_{\Gamma_h^2}) \right) \mathrm{d}s \\
&\quad + \rho_u \int_{\Omega_h^\Gamma}^{5} \frac{\partial \mathbf{u}}{\partial \mathbf{n}_{\Gamma_h^2}} \cdot \frac{\partial \mathbf{v}}{\partial \mathbf{n}_{\Gamma_h^2}}\,\mathrm{d}\mathbf{x}, \quad \mathbf{A} \in \mathbb{R}^{n_\mathbf{A} \times n_\mathbf{A}}, \\
\langle \mathbf{B}\,\vec{\mathbf{u}}, \vec{\mathbf{q}} \rangle &= -\int_{\Gamma_{h/m}^{2\to 1}}^{5} q\ \mathrm{div}_{\Gamma_h^2}\,\mathbf{u}\,\mathrm{d}s, \quad \mathbf{B} \in \mathbb{R}^{n_\mathbf{S} \times n_\mathbf{A}}, \\
\langle \mathbf{M}_0\,\vec{\mathbf{p}}, \vec{\mathbf{q}} \rangle &= \int_{\Gamma_{h/m}^{2\to 1}}^{5} p\,q\,\mathrm{d}s, \quad \mathbf{M}_0 \in \mathbb{R}^{n_\mathbf{S} \times n_\mathbf{S}}, \\
\langle \mathbf{C}_n\,\vec{\mathbf{p}}, \vec{\mathbf{q}} \rangle &= \rho_p \int_{\Omega_h^\Gamma}^{5} \frac{\partial p}{\partial \mathbf{n}_{\Gamma_h^2}} \frac{\partial q}{\partial \mathbf{n}_{\Gamma_h^2}}\,\mathrm{d}\mathbf{x}, \quad \mathbf{C}_n \in \mathbb{R}^{n_\mathbf{S} \times n_\mathbf{S}}, \\
\langle \mathbf{C}_{\text{full}}\,\vec{\mathbf{p}}, \vec{\mathbf{q}} \rangle &= \rho_p \int_{\Omega_h^\Gamma}^{5} \nabla p \cdot \nabla q\,\mathrm{d}\mathbf{x}, \quad \mathbf{C}_{\text{full}} \in \mathbb{R}^{n_\mathbf{S} \times n_\mathbf{S}},
\end{aligned}
\tag{16}
$$

Comments:

- $\int_{\Gamma_{h/m}^{2\to 1}}^{5} \cdot\,\mathrm{d}s$ denotes a composite quadrature rule that is exact for $\bar{P}_h^5(\Gamma_{h/m}^{2\to 1})$, i.e. this quadrature is exact for piecewise polynomials up to degree 5 on each triangular patch $\gamma \in \Gamma_{h/m}^{2\to 1}$,
- $\int_{\Omega_h^\Gamma}^{5} \cdot\,\mathrm{d}\mathbf{x}$ denotes a composite quadrature rule that is exact for $\bar{P}_h^5(\Omega_h^\Gamma)$, i.e. this quadrature is exact for piecewise polynomials up to degree 5 on each tetrahedron $T \in \Omega_h^\Gamma$,
- $E_{s,\,\Gamma_h^2}$ and $\mathrm{div}_{\Gamma_h^2}$ are defined as their continuous analogues with $\mathbf{n}_\Gamma$ in $\mathbf{P}$ replaced with $\mathbf{n}_{\Gamma_h^2}$,
- It is always the case that integrands use $\mathbf{n}_{\Gamma_h^2} \neq \mathbf{n}_{\Gamma_{h/m}^{2\to 1}}$, and the actual domain of integration is $\Gamma_{h/m}^{2\to 1} \neq \Gamma_h^2$,
- $\mathbf{n}_{\Gamma_h^2}$ is defined in (14) and it is not a polynomial even locally, thus quadrature rules are never exact (although for $\mathbf{P}_2 - P_1$ shape functions alone these quadratures are exact).

3.5 Using exact normals in integrands of $\int_{\Gamma_{h/m}^{2\to1}}$ (updated summary)

It was quite easy to update (16) such that the exact normals w.r.t. piecewise linear surface domain of integration are used. `spatch` (section 3.3) has a member function that gives **physical** normals to its triangles straightaway. Thus implementation of updated quadratures boiled down to constructing `GridFunction` object (described in section 3.2) out of these physical normals. Details of the implementation can be found in commit `dacc440`.

Comments:

- It is also easy to extend this approach s.t. $\Gamma_{h/m}^{1} \neq \Gamma_{h/m}^{2\to1}$ is used (please see section 3.1 and then figures 5 and 6). There is no difference between $\Gamma_{h/2}^{2\to1}$ and $\Gamma_{h/2}^{1}$. For $m > 2$, there is no difference between $\Gamma_{h/m}^{2\to1}$ and $\Gamma_{h/m}^{1}$ if $\phi \in P^2$. Right now $\Gamma_{h/m}^{2\to1}$ is implemented.
- It is **not** that easy to use the exact normal w.r.t. piecewise linear surface domain of integration in volume integrals. Our guess is that it shall not be a problem (we can leave $\mathbf{n}_{\Gamma_h^2}$ there),
- It is **not** easy to build $I_h^n(\phi)$ for $n > 2$. It is **not** implemented in DROPS as for now.

As for now, the matrices in (1) can also be assembled as

$$
\begin{aligned}
\langle \mathbf{A}\,\vec{\mathbf{u}}, \vec{\mathbf{v}} \rangle &= \int_{\Gamma_{h/m}^{2\to1}}^{5} \left(E_{s,\,\Gamma_{h/m}^{2\to1}}(\mathbf{u}) : E_{s,\,\Gamma_{h/m}^{2\to1}}(\mathbf{v}) + \mathbf{u}\cdot\mathbf{v} + \tau\,(\mathbf{u}\cdot\mathbf{n}_{\Gamma_h^2})\,(\mathbf{v}\cdot\mathbf{n}_{\Gamma_h^2}) \right) \mathrm{d}s \\
&\quad + \rho_u \int_{\Omega_h^\Gamma}^{5} \frac{\partial \mathbf{u}}{\partial \mathbf{n}_{\Gamma_h^2}} \cdot \frac{\partial \mathbf{v}}{\partial \mathbf{n}_{\Gamma_h^2}}\,\mathrm{d}\mathbf{x}, \quad \mathbf{A} \in \mathbb{R}^{n_\mathbf{A}\times n_\mathbf{A}}, \\
\langle \mathbf{B}\,\vec{\mathbf{u}}, \vec{\mathbf{q}} \rangle &= -\int_{\Gamma_{h/m}^{2\to1}}^{5} q\ \mathrm{div}_{\Gamma_{h/m}^{2\to1}}\,\mathbf{u}\,\mathrm{d}s, \quad \mathbf{B} \in \mathbb{R}^{n_\mathbf{S}\times n_\mathbf{A}}, \\
\langle \mathbf{M}_0\,\vec{\mathbf{p}}, \vec{\mathbf{q}} \rangle &= \int_{\Gamma_{h/m}^{2\to1}}^{5} p\,q\,\mathrm{d}s, \quad \mathbf{M}_0 \in \mathbb{R}^{n_\mathbf{S}\times n_\mathbf{S}}, \\
\langle \mathbf{C}_n\,\vec{\mathbf{p}}, \vec{\mathbf{q}} \rangle &= \rho_p \int_{\Omega_h^\Gamma}^{5} \frac{\partial p}{\partial \mathbf{n}_{\Gamma_h^2}} \frac{\partial q}{\partial \mathbf{n}_{\Gamma_h^2}}\,\mathrm{d}\mathbf{x}, \quad \mathbf{C}_n \in \mathbb{R}^{n_\mathbf{S}\times n_\mathbf{S}}, \\
\langle \mathbf{C}_{\text{full}}\,\vec{\mathbf{p}}, \vec{\mathbf{q}} \rangle &= \rho_p \int_{\Omega_h^\Gamma}^{5} \nabla p \cdot \nabla q\,\mathrm{d}\mathbf{x}, \quad \mathbf{C}_{\text{full}} \in \mathbb{R}^{n_\mathbf{S}\times n_\mathbf{S}},
\end{aligned}
\tag{17}
$$

Notations are as in (16). Note that the "τ-term" uses $\mathbf{n}_{\Gamma_h^2}$. In order to switch between (16) and (17), one modifies JSON input file:

```
// No_Bnd_Condition.json
"Levelset": {
// ...
"NumbOfVirtualSubEdges" : 2,
"UseExactNormals"       : "yes",
// ...
}
```

Here 2 corresponds to $m = 2$, and "yes" corresponds to (17).

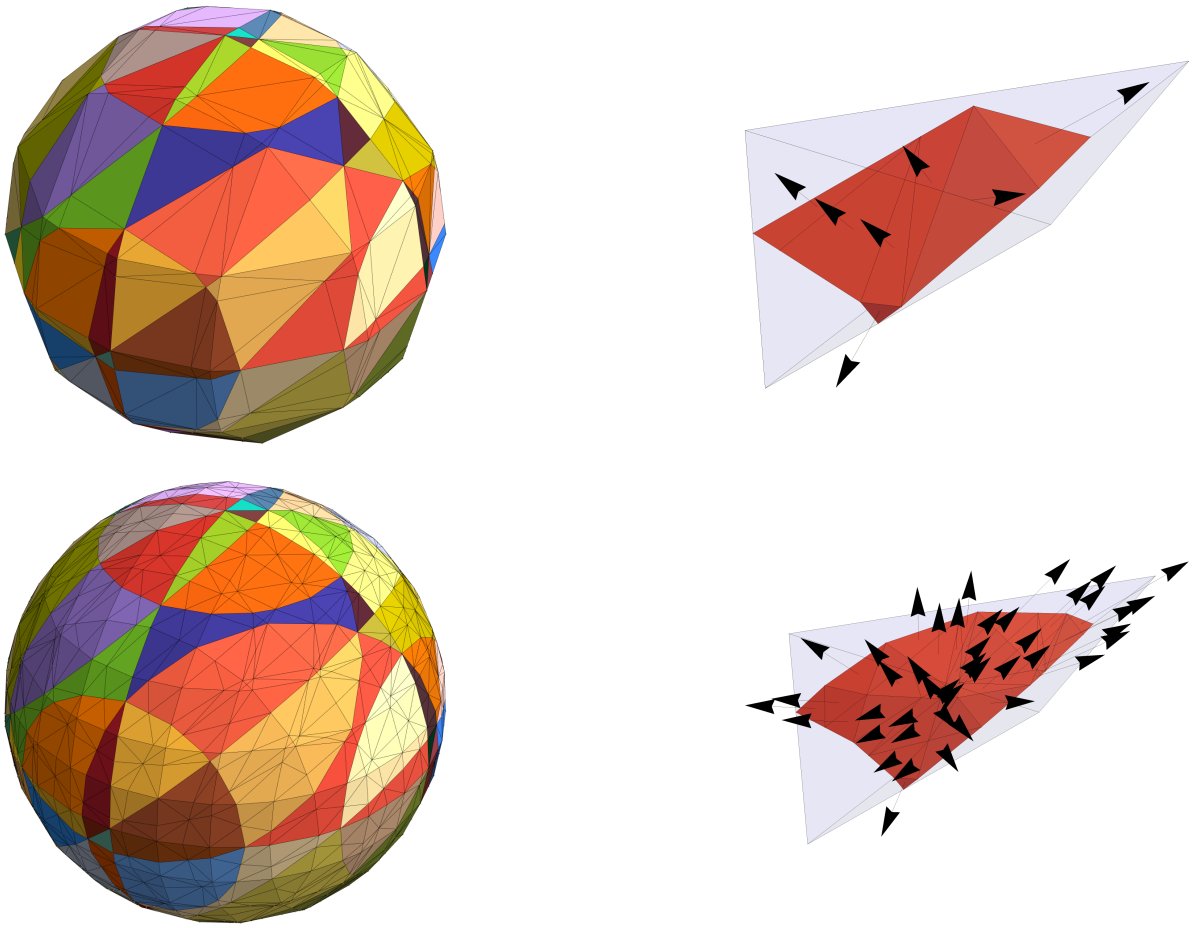


Figure 5: $\Gamma = \Gamma_{\text{sph}}$, $\phi(\mathbf{x}) = \|\mathbf{x}\|^2 - 1$, $h = 8.33 \times 10^{-1}$. Top-left: $\Gamma^{2\to1}_{h/2} = \Gamma^1_{h/2}$ (different color corresponds to a different `spatch` $\gamma \in \Gamma^{2\to1}_{h/2}$ as described in section 3.3). Top-right: a patch $\gamma \in \Gamma^{2\to1}_{h/2}$ and its normals. Bottom-left and bottom-right: same for $\Gamma^{2\to1}_{h/4} = \Gamma^1_{h/4}$. **Note that since $\phi \in P^2$, we have that $\Gamma^{2\to1}_{h/m} = \Gamma^1_{h/m} \to \Gamma$ as $m \to \infty$ independent of h**

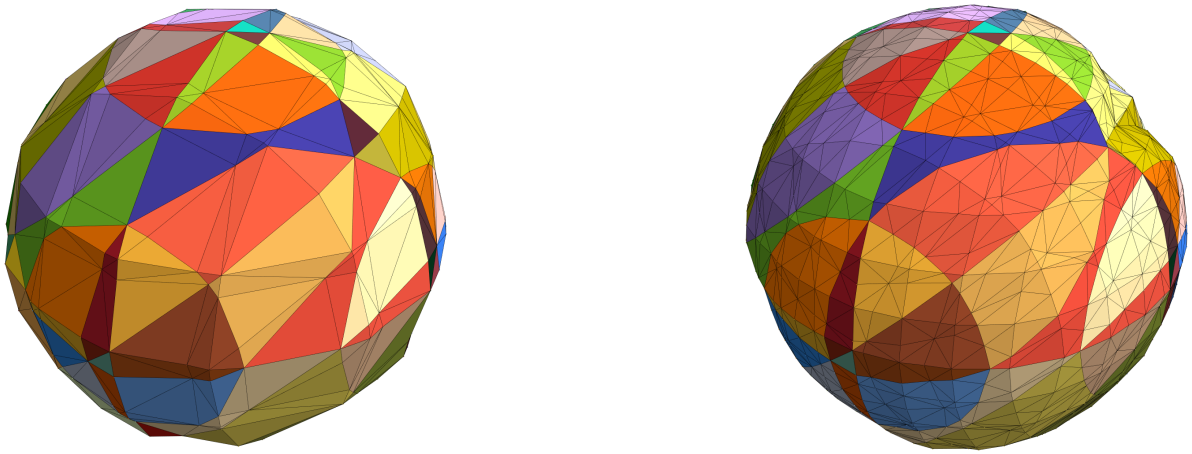


Figure 6: $\Gamma = \Gamma_{\text{sph}}$, $\phi(\mathbf{x}) = \|\mathbf{x}\|^{1/2} - 1$, $h = 8.33 \times 10^{-1}$. Left: $\Gamma^{2\to1}_{h/2} = \Gamma^1_{h/2}$ (different color corresponds to a different `spatch` $\gamma \in \Gamma^{2\to1}_{h/2}$ as described in section 3.3). Right: same for $\Gamma^{2\to1}_{h/4} \neq \Gamma^1_{h/4}$. **Note that since $\phi \notin \bar{P}^2_h$, we have that $\Gamma^{2\to1}_{h/m} \neq \Gamma^1_{h/m}$ for $m > 2$, and $\Gamma^{2\to1}_{h/m} \to \Gamma^2_h \neq \Gamma$ as $m \to \infty$ for fixed h**

3.6 Quadrature rules for the error computation

When we first tried to test convergence for (17) in section 2, we noticed that the $\mathbb{H}^1(\Gamma)$-error of the velocity decays much slower than expected, whereas its $\mathbb{L}^2(\Gamma)$-error behaves as expected. Note that $\mathbb{H}^1(\Gamma)$-error (for e.g. $\mathbf{P}_2 - P_1$ FE) can be computed as $\langle \mathbf{w}, \mathbf{A}_s \mathbf{w} \rangle^{1/2}$, $\mathbf{w} \coloneqq$ vector of d.o.f. corresponding to $\mathbf{P}_h^2$ interpolant $I_h^2(\mathbf{u}) - \mathbf{u}_h$, $\mathbf{A}_s \coloneqq$ matrix corresponding to the first term of $\mathbf{A}$ in (17). Thus the errors are approximated as

$$\begin{aligned}
\|\mathbf{u} - \mathbf{u}_h\|_{\mathbb{H}^1(\Gamma)} &= \|I_h^k(\mathbf{u}) - \mathbf{u}_h\|_{\mathbb{H}^1\left(\Gamma_{h/m}^{2\to 1}\right)} + O(h^k), \\
\|\mathbf{u} - \mathbf{u}_h\|_{\mathbb{L}^2(\Gamma)} &= \|I_h^k(\mathbf{u}) - \mathbf{u}_h\|_{\mathbb{L}^2\left(\Gamma_{h/m}^{2\to 1}\right)} + O(h^{k+1}), \\
\|p - p_h\|_{\mathbb{L}^2(\Gamma)} &= \|I_h^1(p) - p_h\|_{\mathbb{L}^2\left(\Gamma_{h/m}^{2\to 1}\right)} + O(h^2)
\end{aligned}$$

for large enough m. Here $k = 1$ for $\mathbf{P}_1 - P_1$ FEM and $k = 2$ for $\mathbf{P}_2 - P_1$.

Interestingly enough, DROPS implementation did not use the assembled matrices to compute errors (and normals that are *different* from the ones in $\mathbf{A}_s$ were used). We corrected it in commit 68443b0:

```
// surfnavierstokes.cpp
// ...
VectorCL vSolMinusV = vSol.Data - v.Data, pSolMinusP = pSol.Data - p.Data;
auto velL2          = sqrt(dot(v.Data, M.Data * v.Data));
auto velNormalL2    = sqrt(dot(v.Data, S.Data * v.Data));
auto velH1err       = sqrt(dot(vSolMinusV, A.Data * vSolMinusV));
auto velL2err       = sqrt(dot(vSolMinusV, M.Data * vSolMinusV));
auto preL2          = sqrt(dot(p.Data, Schur.Data * p.Data));
auto preL2err       = sqrt(dot(pSolMinusP, Schur.Data * pSolMinusP));
// ...
```

References

[1] M. Olshanskii, A. Quaini, A. Reusken, and V. Yushutin. A finite element method for the surface stokes problem. *SIAM Journal on Scientific Computing*, 40(4):A2492–A2518, 2018.