

# Chapter 1

## Demo problem: A preconditioner for the solution of Navier-Stokes equations with weakly imposed boundary conditions via Lagrange multipliers

The purpose of this tutorial is to show how to use `oomph-lib`'s Lagrange Enforced Flow Navier-Stokes preconditioner. Similarly to the problem considered in the [Steady finite-Reynolds-number flow through an iliac bifurcation](#) tutorial, the outflow boundary of the demo problem (discussed below) are not aligned with any coordinate planes. Parallel outflow is therefore enforced by a Lagrange multiplier method, implemented using `oomph-lib`'s FaceElement framework.

### 1.1 The model problem, theory and preconditioner

We will demonstrate the development and application of the preconditioner using the Poiseuille flow through a unit square domain  $\Omega^{[\alpha]} \in \mathbb{R}^2$  rotated by an arbitrary angle  $\alpha$  (see the figure below). The domain  $\Omega^{[\alpha]}$  is obtained by rotating the discrete points  $(x_1, x_2)$  in the unit square  $\Omega = [0, 1]^2$  by the following transformation

$$R(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}, \quad (1)$$

where  $\alpha$  is the angle of rotation. The figure below show the flow field (velocity vectors and pressure contours) for a unit square domain rotated by an angle of  $\alpha = 30^\circ$  and a Reynolds number of  $Re = 100$ . The flow is driven by a prescribed parabolic boundary condition.

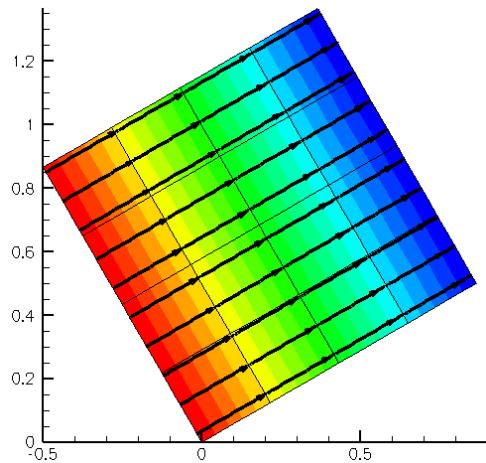


Figure 1.1 Velocity field and pressure

For convenience, we present the boundary conditions for the non-rotated unit square  $\alpha = 0^\circ$ . In order to obtain the boundary conditions for  $\alpha \neq 0^\circ$ , we only have to apply the rotation (1). The flow is driven by imposing a parabolic velocity profile along the inflow boundary  $\Omega_I$ . Along the characteristic boundary,  $\Omega_C$ , the no-slip condition  $u_i = 0$ ,  $i = 1, 2$ , is prescribed. We impose 'parallel outflow' along the outlet  $\Omega_O$  by insisting that

$$\mathbf{u} \cdot \mathbf{t} = 0 \quad \text{on } \Omega_O, \quad (2)$$

where  $\mathbf{t}$  is the tangent vector at each discrete point on the boundary  $\Omega_O$ . We weakly enforce the flow constraint by augmenting the Navier-Stokes momentum residual equation (introduced in the [Unsteady flow in a 2D channel, driven by an applied traction](#) tutorial) with a Lagrange multiplier term so that it becomes

$$r_{il}^u = \int_{\Omega} \left[ Re \left( St \frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} \right) \psi_l + \tau_{ij} \frac{\partial \psi_l}{\partial x_j} \right] d\Omega - \int_{\partial\Omega} \tau_{ij} n_j \psi_l dS + \delta \Pi_{constraint} = 0, \quad (3)$$

where

$$\Pi_{constraint} = \int_{\partial\Omega} \lambda u_i t_i dS, \quad (4)$$

and  $\lambda$  is the Lagrange multiplier. Upon taking the first variation of the constraint with respect to the unknown velocity and the Lagrange multiplier, the residual form of the constrained momentum equation is

$$r_{il}^u = \int_{\Omega} \left[ Re \left( St \frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} \right) \psi_l + \tau_{ij} \frac{\partial \psi_l}{\partial x_j} \right] d\Omega - \int_{\partial\Omega} \tau_{ij} n_j \psi_l dS + \int_{\partial\Omega} \lambda \psi_l t_i = 0. \quad (5)$$

The weak formulation of (2) is simply

$$r_i^\lambda = \int_{\Omega} u_i t_i \psi^\lambda dS = 0, \quad (6)$$

where  $\psi^\lambda$  is a suitable basis function. Equation (5) reveals that the Lagrange multipliers act as the (negative) tangential traction ( $\lambda = -\mathbf{n}^T \tau \mathbf{t}$ ) that enforce the parallel flow across the boundary  $\partial\Omega_O$ . We discretise this constraint by attaching `ImposeParallelOutflowElements` to the boundaries of the "bulk" Navier-Stokes elements that are adjacent to  $\partial\Omega_O$  as shown in the [Steady finite-Reynolds-number flow through an iliac bifurcation](#) tutorial, also see the [Deformation of a solid by a prescribed boundary motion](#) tutorial which employs a similar technique used to enforce prescribed boundary displacements in solid mechanics problems. We discretise the Navier-Stokes equations using `oomph-lib`'s `QTaylorHoodElements`, see the [2D Driven Cavity Problem](#) tutorial for more information.

The discretised problem therefore contains the following types of discrete unknowns:

- The fluid degrees of freedom (velocity and pressure).
  
  
  
  
  
  
- The nodal values representing the components of the (vector-valued) Lagrange multipliers. These only exist for the nodes on  $\partial\Omega_O$ . (The nodes are re-sized to accommodate the additional unknowns when the `ImposeParallelOutflowElements` are attached to the bulk elements.)

The preconditioner requires a further sub-division of these degrees of freedom into the following categories:

- the unconstrained velocity in the x-direction
- the unconstrained velocity in the y-direction
- [the unconstrained velocity in the z-direction (only in 3D)]
- the constrained velocity in the x-direction
- the constrained velocity in the y-direction
- [the constrained velocity in the z-direction (only in 3D)]
- the Lagrange multiplier at the constrained nodes
- [the other Lagrange multiplier at the constrained nodes (only in 3D)].

For a 2D problem, the linear system to be solved in the course of the Newton iteration can then be (formally) re-ordered into the following block structure:

$$\left[ \begin{array}{ccccc|c} F_{xx} & F_{x\bar{x}} & F_{xy} & F_{x\bar{y}} & B_x^T & M_x \\ F_{\bar{x}x} & F_{\bar{x}\bar{x}} & F_{\bar{x}y} & F_{\bar{x}\bar{y}} & B_{\bar{x}}^T & \\ F_{yx} & F_{y\bar{x}} & F_{yy} & F_{y\bar{y}} & B_y^T & M_y \\ F_{\bar{y}x} & F_{\bar{y}\bar{x}} & F_{\bar{y}y} & F_{\bar{y}\bar{y}} & B_{\bar{y}}^T & \\ \hline B_x & B_{\bar{x}} & B_y & B_{\bar{y}} & & \\ M_x & M_{\bar{x}} & M_y & M_{\bar{y}} & & \end{array} \right] \left[ \begin{array}{c} \Delta \mathbf{U}_x \\ \Delta \bar{\mathbf{U}}_x \\ \Delta \mathbf{U}_y \\ \Delta \bar{\mathbf{U}}_y \\ \Delta \mathbf{P} \\ \Delta \Lambda \end{array} \right] = - \left[ \begin{array}{c} \mathbf{r}_x \\ \mathbf{r}_{\bar{x}} \\ \mathbf{r}_y \\ \mathbf{r}_{\bar{y}} \\ \mathbf{r}_p \\ \mathbf{r}_{\Lambda} \end{array} \right]. \quad (7)$$

Here the vectors  $\mathbf{U}_x$ ,  $\mathbf{U}_y$ ,  $\mathbf{P}$  and  $\Lambda$  contain the  $x$  and  $y$  components of the velocity unknowns, the pressure unknowns and Lagrange multipliers unknowns, respectively. The overbars identify the unknown nodal positions that are constrained by the Lagrange multiplier. The matrices  $M_x$  and  $M_y$  are mass-like matrices whose entries are formed from products of the basis functions multiplied by a component of the tangent vector at each discrete point on  $\partial\Omega_O$ , for example,  $[M_x]_{ij} = \int_{\partial\Omega_O} t_x \psi_i \psi_j dS$ . Denote

$$J_{NS} = \left[ \begin{array}{ccccc} F_{xx} & F_{x\bar{x}} & F_{xy} & F_{x\bar{y}} & B_x^T \\ F_{\bar{x}x} & F_{\bar{x}\bar{x}} & F_{\bar{x}y} & F_{\bar{x}\bar{y}} & B_{\bar{x}}^T \\ F_{yx} & F_{y\bar{x}} & F_{yy} & F_{y\bar{y}} & B_y^T \\ F_{\bar{y}x} & F_{\bar{y}\bar{x}} & F_{\bar{y}y} & F_{\bar{y}\bar{y}} & B_{\bar{y}}^T \\ B_x & B_{\bar{x}} & B_y & B_{\bar{y}} & \end{array} \right], \quad L = \left[ \begin{array}{cc} M_x & M_{\bar{x}} \end{array} \right], \quad \Delta \mathbf{X}_{NS} = \left[ \begin{array}{c} \Delta \mathbf{U}_x \\ \Delta \bar{\mathbf{U}}_x \\ \Delta \mathbf{U}_y \\ \Delta \bar{\mathbf{U}}_y \\ \Delta \mathbf{P} \end{array} \right], \quad \text{and} \quad \mathbf{r}_{NS} = \left[ \begin{array}{c} \mathbf{r}_x \\ \mathbf{r}_{\bar{x}} \\ \mathbf{r}_y \\ \mathbf{r}_{\bar{y}} \\ \mathbf{r}_p \end{array} \right].$$

Then we can re-write (7) as

$$\left[ \begin{array}{cc} J_{NS} & L^T \\ L & \end{array} \right] \left[ \begin{array}{c} \Delta \mathbf{X}_{NS} \\ \Delta \Lambda \end{array} \right] = - \left[ \begin{array}{c} \mathbf{r}_{NS} \\ \mathbf{r}_{\Lambda} \end{array} \right]. \quad (8)$$

We have shown that

$$P = \left[ \begin{array}{cc} J_{NS} + L^T W^{-1} L & \\ & W \end{array} \right], \quad (9)$$

where  $W = \frac{1}{\sigma} LL^T$  is an optimal preconditioner for the linear system (8) if we set  $\sigma = \|F\|_\infty$  where  $F$  is the compound  $4 \times 4$  top-left block

$$F = \left[ \begin{array}{cccc} F_{xx} & F_{x\bar{x}} & F_{xy} & F_{x\bar{y}} \\ F_{\bar{x}x} & F_{\bar{x}\bar{x}} & F_{\bar{x}y} & F_{\bar{x}\bar{y}} \\ F_{yx} & F_{y\bar{x}} & F_{yy} & F_{y\bar{y}} \\ F_{\bar{y}x} & F_{\bar{y}\bar{x}} & F_{\bar{y}y} & F_{\bar{y}\bar{y}} \end{array} \right]$$

in the Jacobian matrix. Application of the preconditioner  $P$  requires the repeated solution of linear systems involving the diagonal blocks  $J_{NS} + L^T W^{-1} L$  and  $W$ . The matrix  $W^{-1} = \sigma(LL^T)^{-1} = \sigma(M_x^2 + M_y^2)^{-1}$  is dense and will cause the addition of dense sub-matrices to the Jacobian matrix:

$$\sigma L^T (LL^T)^{-1} L = \sigma \left[ \begin{array}{ccccc} \mathcal{O} & \mathcal{O} & \mathcal{O} & \mathcal{O} & \mathcal{O} \\ \mathcal{O} & M_x(M_x^2 + M_y^2)^{-1} M_x & \mathcal{O} & M_x(M_x^2 + M_y^2)^{-1} M_y & \mathcal{O} \\ \mathcal{O} & \mathcal{O} & \mathcal{O} & \mathcal{O} & \mathcal{O} \\ \mathcal{O} & M_y(M_x^2 + M_y^2)^{-1} M_x & \mathcal{O} & M_y(M_x^2 + M_y^2)^{-1} M_y & \mathcal{O} \\ \mathcal{O} & \mathcal{O} & \mathcal{O} & \mathcal{O} & \mathcal{O} \end{array} \right].$$

Numerical experiments show that an efficient implementation can be obtained by replacing  $W$  by its diagonal approximation  $\widehat{W} = \text{diag}(M_x^2 + M_y^2)$ . Then the inversion of  $W$  is straight forward and the addition of  $L^T \widehat{W}^{-1} L$  to the Jacobian does not significantly increase the number of non-zero entries in the matrix  $J_{NS}$ . Denote the efficient implementation by

$$\tilde{P} = \left[ \begin{array}{cc} \tilde{J}_{NS} & \\ & \widehat{W} \end{array} \right], \quad (10)$$

where  $\tilde{J}_{NS} = J_{NS} + L^T \widehat{W}^{-1} L$  is the augmented Navier-Stokes Jacobian matrix. In our implementation of the preconditioner, the linear system involving  $\tilde{J}_{NS}$  can either be solved "exactly", using SuperLU (in its incarnation as an exact preconditioner; this is the default) or by any other Preconditioner (interpreted as an "inexact solver") specified via the access function

```
LagrangeEnforcedFlowPreconditioner::set_navier_stokes_preconditioner(...)
```

Numerical experiments show that a nearly optimal preconditioner is obtained by replacing the solution of the linear system involving the augmented Navier-Stokes Jacobian  $\tilde{J}_{NS}$  by an application of Elman, Silvester and Wathen's

[Least-Squares Commutator \(LSC\) preconditioner](#), and by replacing the remaining block-solves within these preconditioners by a small number of AMG cycles.

With these approximations, the computational cost of one application of  $\tilde{P}$  is linear in the number of unknowns. The optimality of the preconditioner can therefore be assessed by demonstrating that the number of GMRES iterations remains constant under mesh refinement.

---

## 1.2 Demo driver and use of the preconditioner

To demonstrate how to use the preconditioner, here are the relevant extracts from the driver code [two\\_<→d\\_tilted\\_square.cc](#) which solves the model problem described above. As explained in the [Linear Solvers Tutorial](#) switching to an iterative linear solver is typically performed in the `Problem` constructor and involves a few straightforward steps:

### 1. Create an instance of the `IterativeLinearSolver` and pass it to the `Problem`

In our problem, we choose right preconditioned GMRES as the linear solver:

```
// Create oomph-lib iterative linear solver.
IterativeLinearSolver* solver_pt = new GMRES<CRDoubleMatrix>;

// We use RHS preconditioning. Note that by default,
// left hand preconditioning is used.
static_cast<GMRES<CRDoubleMatrix>*>(solver_pt)
->set_preconditioner_RHS();

// Store the solver pointer.
Solver_pt = solver_pt;
```

### 2. Create an instance of the Preconditioner and give it access to the meshes

The `LagrangeEnforceFlowPreconditioner` takes a pointer of meshes. It is important that the bulk mesh is in position 0 :

```
// Create the preconditioner
LagrangeEnforcedFlowPreconditioner* lgr_prec_pt
= new LagrangeEnforcedFlowPreconditioner;

// Create the vector of mesh pointers!
Vector<Mesh*> mesh_pt;
mesh_pt.resize(2,0);
mesh_pt[0] = Bulk_mesh_pt;
mesh_pt[1] = Surface_mesh_P_pt;

lgr_prec_pt->set_meshes(mesh_pt);
```

By default, `SuperLUPreconditioner` is used for all subsidiary block solves. To use the LSC preconditioner to approximately solve the sub-block system involving the momentum block, we invoke the following:

```
// Create the NS LSC preconditioner.
lsc_prec_pt = new NavierStokesSchurComplementPreconditioner(this);
lsc_prec_pt->set_navier_stokes_mesh(Bulk_mesh_pt);
lsc_prec_pt->use_lsc();
lgr_prec_pt->set_navier_stokes_preconditioner(lsc_prec_pt);
```

The LSC preconditioner is discussed in [another tutorial](#).

### 3. Pass the preconditioner to the solver, and the solver to the problem

```
// Pass the preconditioner to the solver.
Solver_pt->preconditioner_pt() = lgr_prec_pt;

// Pass the solver to the problem.
this->linear_solver_pt() = Solver_pt;
```

## 1.3 Source files for this tutorial

- The source files for this tutorial are located in the directory:

```
demo_drivers/navier_stokes/lagrange_enforced_flow_preconditioner
```

- The driver code is:

```
demo_drivers/navier_stokes/lagrange_enforced_flow_preconditioner/two_←  
d_tilted_square.cc
```

---

## 1.4 PDF file

A [pdf version](#) of this document is available. \