A Space-Time Multigrid Method for Optimal Flow Control

Michael Hinze, Michael Köster and Stefan Turek

Abstract. We present a hierarchical solution concept for optimization problems governed by the time-dependent Navier–Stokes system. Discretisation is carried out with finite elements in space and a one-step-θ-scheme in time to generate a space-time mesh hierarchy. By combining a Newton solver for the treatment of the nonlinearity with a space-time multigrid solver for linear subproblems, we obtain a robust solver whose convergence behaviour is independent of the refinement level of the discrete problem. A set of numerical examples analyses the solver behaviour for various problem settings with respect to efficiency and robustness of this approach.

Mathematics Subject Classification (2000). 35Q30, 49K20, 49M05, 49M15, 49M29, 65M55, 65M60, 76D05, 76D55.

Keywords. distributed control, finite elements, time-dependent Navier–Stokes, Newton, space-time multigrid, optimal control.

1. Introduction

Active flow control plays a central role in many practical applications such as e.g. control of crystal growth processes [9, 16, 15, 17], where the flow in the melt has a significant impact on the quality of the crystal. Optimal control of the flow by electro-magnetic fields and/or boundary temperatures leads to optimisation problems with PDE constraints, which are frequently governed by the time-dependent Navier-Stokes system.

The mathematical formulation is a minimisation problem with PDE constraints. By exploiting the special structure of the first order necessary optimality conditions, the so called Karush-Kuhn-Tucker (KKT)-system, we are able to develop a hierarchical solution approach for the optimisation of the Stokes– and Navier–Stokes equations which satisfies

$$\frac{\text{effort for optimisation}}{\text{effort for simulation}} \leq C, \tag{1}$$

for a constant $C > 0$ of moderate size, independent of the refinement level. Tests show a factor of $20 - 30$ for the Navier–Stokes equations. Here, the effort for the simulation is assumed to be optimal in that sense, that a solver needs $O(N)$ operations, N denoting the total number of unknowns for a given computational mesh in space and time. This can be achieved by utilising appropriate multigrid techniques for the linear subproblems in space. Because of (1), the developed solution approach for the optimal control problem has also complexity $O(N)$; this is achieved by a combination of a space-time Newton approach for the nonlinearity and a space-time multigrid approach for linear subproblems. The complexity of this algorithm distinguishes our method from adjoint-based steepest descent methods used to solve optimisation problems in many practical applications, which in general do not satisfy this complexity requirement. A related approach can be found, e.g. in [4] where multigrid methods for the numerical solution of optimal control problems for parabolic PDEs are developed based on Finite Difference techniques for the discretisation. In [6] a space-time multigrid method for the corresponding *integral equation approach* of [10] is developed, compare also [8].

The paper is organised as follows: In Section 2 we describe the discretisation of a flow control problem and give an introduction to the ingredients needed to design a multigrid solver. The discretisation is carried out with finite elements in space and finite differences in time. In Section 3 we propose the basic algorithms that are necessary to construct our multigrid solver for linear and a Newton solver for nonlinear problems. Finally, Section 4 is devoted to numerical examples which we present to confirm the predicted behaviour.

2. Problem formulation and discretisation

We consider the optimal control problem

$$J(y,u) := \frac{1}{2}||y-z||^2_{L^2(Q)} + \frac{\alpha}{2}||u||^2_{L^2(Q)} + \frac{\gamma}{2}||y(T)-z(T)||^2_{L^2(\Omega)} \quad \longrightarrow \quad \text{min!} \quad (2)$$

$$\begin{aligned} \text{s.t.} \qquad y_t - \nu\Delta y + y\nabla y + \nabla p &= u && \text{in } Q, \\ -\text{div } y &= 0 && \text{in } Q, \\ y(0,\cdot) &= y^0 && \text{in } \Omega, \\ y &= g && \text{at } \Sigma. \end{aligned}$$

Here, $\Omega \subset \mathbb{R}^d$ $(d = 2, 3)$ denotes an open bounded domain, $\Gamma := \partial\Omega$, $T > 0$ defines the time horizon, and $Q = (0,T)\times\Omega$ denotes the corresponding space-time cylinder with space-time boundary $\Sigma := (0,T)\times\Gamma$. The function $g : \Sigma \to \mathbb{R}^d$ specifies some Dirichlet boundary conditions, u denotes the control, y the velocity vector, p the pressure and z a given target velocity field for y. Finally, $\gamma \geq 0$, $\alpha > 0$ denote constants. For simplicity, we do not assume any restrictions on the control u.

The first order necessary optimality conditions are then given through the so called *Karush-Kuhn-Tucker* system

$$\begin{aligned}
y_t - \nu\Delta y + y\nabla y + \nabla p &= u && \text{in } Q\\
-\text{div } y &= 0 && \text{in } Q\\
y(t,\cdot) &= g(t,\cdot) && \text{on } \Gamma \text{ for all } t \in [0,T]\\
y(0,\cdot) &= y^0 && \text{in } \Omega
\end{aligned}$$

$$\begin{aligned}
-\lambda_t - \nu\Delta\lambda - y\nabla\lambda + (\nabla y)^t\lambda + \nabla\xi &= y - z && \text{in } Q\\
-\text{div } \lambda &= 0 && \text{in } Q\\
\lambda(t,\cdot) &= 0 && \text{at } \Gamma \text{ for all } t \in [0,T]\\
\lambda(T) &= \gamma(y(T) - z(T)) && \text{in } \Omega
\end{aligned}$$

$$u = -\tfrac{1}{\alpha}\lambda,$$

where λ denotes the dual velocity and ξ the dual pressure. We eliminate u in the KKT system, and (ignoring boundary conditions at the moment), we obtain

$$\begin{aligned}
y_t - \nu\Delta y + y\nabla y + \nabla p &= -\frac{1}{\alpha}\lambda, \qquad (3)\\
-\text{div } y &= 0,\\
y(0,\cdot) &= y_0,
\end{aligned}$$

$$\begin{aligned}
-\lambda_t - \nu\Delta\lambda - y\nabla\lambda + (\nabla y)^t\lambda + \nabla\xi &= y - z, \qquad (4)\\
-\text{div } \lambda &= 0,\\
\lambda(T) &= \gamma(y(T) - z(T))
\end{aligned}$$

where we call (3) the *primal* and (4) the *dual* equation.

Coupled discretisation in time. For the discretisation of the system, we follow the approach *first optimise, then discretise*: We semi-discretise the KKT-system in time. For stability reasons (cf. [22]) we prefer implicit time stepping techniques. This allows us to be able to choose the timestep size only depending on the accuracy demands, independent of any stability constraints. The approach is demonstrated on the standard 1st order backward Euler scheme as a representative of implicit schemes. Higher order schemes like Crank-Nicolson are possible as well but lead to a more complicated matrix structure (with a couple of nonlinear terms also on the offdiagonals) and, depending on the time discretisation, some less straight-forward time prolongation/restriction.

The time discretisation of (3) yields

$$\begin{aligned}
\frac{y_{k+1} - y_k}{\Delta t} - \nu\Delta y_{k+1} + y_{k+1}\nabla y_{k+1} + \nabla p_{k+1} &= -\frac{1}{\alpha}\lambda_{k+1} \qquad (5)\\
-\text{div } y_{k+1} &= 0\\
y_0 &= y^0
\end{aligned}$$

where $N \in \mathbb{N}$, $k = 0, ..., N-1$ and $\Delta t = 1/N$. To (3), (4) we apply the discretisation recipe from [3]. For this purpose, we define the following operators: $\mathcal{A}v := -\nu\Delta v$,

$\mathcal{I}v := v$, $\mathcal{G}q := \nabla q$, $\mathcal{D}v := -\text{div } v$, $\mathcal{K}_n v := \mathcal{K}(y_n)v := (y_n\nabla)v$, $\overline{\mathcal{K}}_n v := \overline{\mathcal{K}}(y_n)v := (v\nabla)y_n$ and $\mathcal{C}_n v := \mathcal{C}(y_n)v := \mathcal{A}v + \mathcal{K}(y_n)v$ for all velocity vectors v and and pressure functions q in space, $n \in \mathbb{N}$.

As the initial solution y^0 may not be solenoidal, we realise the initial condition $y_0 = y^0$ by the following solenoidal projection,

$$\begin{aligned} \frac{1}{\Delta t}y_0 - \nu\Delta y_0 + y_0\nabla y_0 + \nabla p_0 &= \frac{1}{\Delta t}y^0 - \nu\Delta y^0 + y^0\nabla y^0 \\ -\text{div } y_0 &= 0. \end{aligned}$$

This projection operator uses the same operations like the backward Euler scheme which allows for a more consistent notation. Using $x := (y_0, p_0, y_1, p_1, ..., y_N, p_N)$, this yields the nonlinear system of the primal equation,

$$\begin{aligned} \mathcal{H}x &:= \mathcal{H}(x)x \\ &= \left(\begin{array}{c|c|c|c} \frac{\mathcal{I}}{\Delta t} + \mathcal{C}_0 \quad \mathcal{G} & & & \\ \mathcal{D} & & & \\ \hline -\frac{\mathcal{I}}{\Delta t} & \frac{\mathcal{I}}{\Delta t} + \mathcal{C}_1 \quad \mathcal{G} & & \\ & \mathcal{D} & & \\ \hline & \ddots & \ddots \quad \ddots & \\ \hline & & -\frac{\mathcal{I}}{\Delta t} & \frac{\mathcal{I}}{\Delta t} + \mathcal{C}_N \quad \mathcal{G} \\ & & & \mathcal{D} \end{array}\right) \left(\begin{array}{c} y_0 \\ p_0 \\ y_1 \\ p_1 \\ \vdots \\ y_N \\ p_N \end{array}\right) \\ &= \left((\frac{\mathcal{I}}{\Delta t} + \mathcal{C}_0)y^0, 0, -\frac{\lambda_1}{\alpha}, 0, ..., -\frac{\lambda_N}{\alpha}, 0, \right) \end{aligned}$$

which is equivalent to (5) if y^0 is solenoidal. In the second step, we focus on the Fréchet derivative of the Navier–Stokes equations. For a vector $(\bar{y}, \bar{p})$ the Fréchet derivative in (y, p) reads

$$\mathcal{F}(y,p)\left(\begin{array}{c} \bar{y} \\ \bar{p} \end{array}\right) := \left(\begin{array}{c} \bar{y}_t - \nu\Delta\bar{y} + (\bar{y}\nabla y + y\nabla\bar{y}) + \nabla\bar{p} \\ -\text{div } \bar{y} \end{array}\right).$$

We again carry out the time discretisation as above. For vectors $x := (y_0, p_0, y_1, p_1, ..., y_N, p_N)$ and $\bar{x} := (\bar{y}_0, \bar{p}_0, \bar{y}_1, \bar{p}_1, ..., \bar{y}_N, \bar{p}_N)$ this results in the scheme

$$\begin{aligned} \mathcal{M}\bar{x} &:= \mathcal{M}(x)\bar{x} \\ &= \left(\begin{array}{c|c|c|c} \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_0 \quad \mathcal{G} & & & \\ \mathcal{D} & & & \\ \hline -\frac{\mathcal{I}}{\Delta t} & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_1 \quad \mathcal{G} & & \\ & \mathcal{D} & & \\ \hline & \ddots & \ddots \quad \ddots & \\ \hline & & -\frac{\mathcal{I}}{\Delta t} & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_N \quad \mathcal{G} \\ & & & \mathcal{D} \end{array}\right) \left(\begin{array}{c} \bar{y}_0 \\ \bar{p}_0 \\ \bar{y}_1 \\ \bar{p}_1 \\ \vdots \\ \bar{y}_N \\ \bar{p}_N \end{array}\right) \end{aligned}$$

with the additional operator $\mathcal{N}_n := \mathcal{N}(y_n) := \mathcal{A} + \mathcal{K}(y_n) + \overline{\mathcal{K}}(y_n)$. The time discretisation of the dual equation corresponding to $\mathcal{H}$ is now defined as the adjoint

$\mathcal{M}^*$ of $\mathcal{M}$,

$$(\mathcal{M}\bar{x}, \lambda) = (\bar{x}, \mathcal{M}^*\lambda),$$

where $\lambda := (\lambda_0, \xi_0, \lambda_1, \xi_1, ..., \lambda_N, \xi_N)$. With $\mathcal{N}_n^* := \mathcal{N}^*(y_n) = \mathcal{A} - \mathcal{K}(y_n) + \overline{\mathcal{K}}^*(y_n)$, $\overline{\mathcal{K}}^*(y_n)v = (\nabla y)^t v$ for all velocity vectors v, this reads

$$\begin{aligned}
\mathcal{M}^*\lambda &= \mathcal{M}^*(x)\lambda \\
&= \left(\begin{array}{c|c|c|c}
\frac{\mathcal{I}}{\Delta t} + \mathcal{N}_0^* \quad \mathcal{G} & -\frac{\mathcal{I}}{\Delta t} & & \\
\mathcal{D} & & & \\
\hline
 & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_1^* \quad \mathcal{G} & -\frac{\mathcal{I}}{\Delta t} & \\
 & \mathcal{D} & & \\
\hline
 & & \ddots \quad \ddots & \ddots \\
\hline
 & & & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_N^* \quad \mathcal{G} \\
 & & & \mathcal{D}
\end{array}\right) \lambda \\
&= \Big(y_0 - z_0, 0,\ y_1 - z_1, 0,\ ..., (1 + \frac{\gamma}{\Delta t})(y_N - z_N), 0\Big)^T
\end{aligned}$$

where the right hand side and terminal condition is chosen in such a way that the *optimise-then-discretise* approach we are using here commutes with the *discretise-then-optimise* approach. This corresponds to the time discretisation scheme

$$\begin{aligned}
\frac{\lambda_k - \lambda_{k+1}}{\Delta t} - \nu\Delta\lambda_k - y_k\nabla\lambda_k + (\nabla y_k)^t\lambda_k + \nabla\xi_k &= y_k - z_k \qquad (6)\\
-\text{div } \lambda_k &= 0 \\
\lambda_N &= \gamma(y_N - z_N).
\end{aligned}$$

of (4). Here we have used $\mathcal{D}^* = \mathcal{G}$ and $\mathcal{A}^* = \mathcal{A}$. Now let us define $w_n := (y_n, p_n, \lambda_n, \xi_n)$ and

$$w := (w_0, w_1, ...) := (y_0, \lambda_0, p_0, \xi_0, y_1, \lambda_1, p_1, \xi_1, y_2, \lambda_2, p_2, \xi_2, ...).$$

After shifting the terms with λ_{k+1} and y_k in (5) and (6) from the right hand side to the left hand side and mixing the two matrices stemming from $\mathcal{H}$ and $\mathcal{M}^*$, we obtain a semi-discrete system

$$G(w)w = f. \qquad (7)$$

The right hand side is given by

$$\begin{aligned}
f = \Big(&\underbrace{(\mathcal{I}/\Delta t + \mathcal{C}_0)y^0, -z_0, 0, 0}, \underbrace{0, -z_1, 0, 0}, \\
&..., \underbrace{0, -z_{N-1}, 0, 0}, \underbrace{0, -(1 + \gamma/\Delta t)z_N, 0, 0} \Big)
\end{aligned}$$

and the operator reads

$$G = G(w) = \left(\begin{array}{c|c|c|c|c} \mathcal{G}_0 & \hat{\mathcal{I}}_0 & & & \\ \hline \tilde{\mathcal{I}}_1 & \mathcal{G}_1 & \hat{\mathcal{I}}_1 & & \\ \hline & \tilde{\mathcal{I}}_2 & \mathcal{G}_2 & \hat{\mathcal{I}}_2 & \\ \hline & & \ddots & \ddots & \ddots \\ \hline & & & \tilde{\mathcal{I}}_N & \mathcal{G}_N \end{array}\right) \tag{8}$$

with

$$\mathcal{G}_0 = \begin{pmatrix} \frac{\mathcal{I}}{\Delta t} + \mathcal{C}_0 & 0 & \mathcal{G} & 0 \\ -\mathcal{I} & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_0^* & 0 & \mathcal{G} \\ \mathcal{D} & 0 & 0 & 0 \\ 0 & \mathcal{D} & 0 & 0 \end{pmatrix},\ \mathcal{G}_i = \begin{pmatrix} \frac{\mathcal{I}}{\Delta t} + \mathcal{C}_i & \frac{\mathcal{I}}{\alpha} & \mathcal{G} & 0 \\ -\mathcal{I} & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_i^* & 0 & \mathcal{G} \\ \mathcal{D} & 0 & 0 & 0 \\ 0 & \mathcal{D} & 0 & 0 \end{pmatrix}$$

for $i = 1, ..., N-1$,

$$\tilde{\mathcal{I}}_{i+1} = \begin{pmatrix} -\frac{\mathcal{I}}{\Delta t} & & & \\ & 0 & & \\ & & 0 & \\ & & & 0 \end{pmatrix},\ \hat{\mathcal{I}}_i = \begin{pmatrix} 0 & & & \\ & -\frac{\mathcal{I}}{\Delta t} & & \\ & & 0 & \\ & & & 0 \end{pmatrix}$$

for $i = 0, ..., N-1$ and

$$\mathcal{G}_N = \begin{pmatrix} \frac{\mathcal{I}}{\Delta t} + \mathcal{C}_N & \frac{\mathcal{I}}{\alpha} & \mathcal{G} & 0 \\ -(1 + \frac{\gamma}{\Delta t})\mathcal{I} & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_N^* & 0 & \mathcal{G} \\ \mathcal{D} & 0 & 0 & 0 \\ 0 & \mathcal{D} & 0 & 0 \end{pmatrix}.$$

At this point, we discretise in space with a finite element approach. The fully discrete version of the KKT system is defined by replaceing the operators $\mathcal{I}$, $\mathcal{A}$, $\mathcal{D}$, ... by their finite element versions $\mathcal{I}^h$, $\mathcal{A}^h$, $\mathcal{D}^h$, ... and by incorporating boundary conditions into the right hand side f. We finally end up with the nonlinear system

$$G^h(w^h)w^h = f^h \tag{9}$$

with the vector $w^h := (w_0^h, w_1^h, ...)$ and $w_n^h := (y_n^h,\ \lambda_n^h,\ p_n^h,\ \xi_n^h)$. Note that the system matrix is a block tridiagonal matrix of the form

$$G^h = G^h(w_h) = \left(\begin{array}{c|c|c|c} G_0 & \hat{M}_0 & & \\ \hline \tilde{M}_1 & G_1 & \hat{M}_1 & \\ \hline & \ddots & \ddots & \ddots \\ \hline & & \tilde{M}_N & G_N \end{array}\right) \tag{10}$$

where $N \in \mathbb{N}$ denotes the number of timesteps. This way, the solver for optimal control problem reduces to a solver for a sparse block tridiagonal system where the diagonal blocks $G_n = G_n(w_h)$ correspond to the timesteps of the fully coupled KKT system. This system does not have to be set up in memory in its complete

form: Utilising defect correction algorithms reduces the solution process to a sequence of matrix vector multiplications in space and time. A matrix-vector multiplication of a solution w^h with the space-time matrix G^h on the other hand reduces to $3N+1$ local matrix-vector multiplication sequences, one in each timestep with subsequent $\tilde{M}_n$, G_n and $\hat{M}_n$.

Discretisation of the Newton system associated to (7). The Newton algorithm in space and time can be written in defect correction form as follows:

$$w_{i+1} := w_i + F(w_i)^{-1}(f - G(w_i)w_i), \qquad i \in \mathbb{N}$$

with $F(w)$ being the Frechét derivative of the operator $G(w)w$ which is given by the Newton matrix

$$F(w) = \left(\begin{array}{c|c|c|c|c} \mathcal{F}_0 & \hat{\mathcal{I}}_0 & & & \\ \hline \tilde{\mathcal{I}}_1 & \mathcal{F}_1 & \hat{\mathcal{I}}_1 & & \\ \hline & \tilde{\mathcal{I}}_2 & \mathcal{F}_2 & \hat{\mathcal{I}}_2 & \\ \hline & & \ddots & \ddots & \ddots \\ \hline & & & \tilde{\mathcal{I}}_N & \mathcal{F}_N \end{array}\right)$$

with

$$\mathcal{F}_0 = \begin{pmatrix} \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_0 & 0 & \mathcal{G} & 0 \\ -\mathcal{I} + \mathcal{R}_0 & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_0^* & 0 & \mathcal{G} \\ \mathcal{D} & 0 & 0 & 0 \\ 0 & \mathcal{D} & 0 & 0 \end{pmatrix}, \; \mathcal{F}_i = \begin{pmatrix} \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_i & \frac{1}{\alpha}\mathcal{I} & \mathcal{G} & 0 \\ -\mathcal{I} + \mathcal{R}_i & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_i^* & 0 & \mathcal{G} \\ \mathcal{D} & 0 & 0 & 0 \\ 0 & \mathcal{D} & 0 & 0 \end{pmatrix}$$

for $i = 1, ..., N-1$ and

$$\mathcal{F}_N = \begin{pmatrix} \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_N & \frac{1}{\alpha}\mathcal{I} & \mathcal{G} & 0 \\ -(1+\frac{\gamma}{\Delta t})\mathcal{I} + \mathcal{R}_N & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_N^* & 0 & \mathcal{G} \\ \mathcal{D} & 0 & 0 & 0 \\ 0 & \mathcal{D} & 0 & 0 \end{pmatrix}.$$

Here, we use the additional operator $\mathcal{R}_n v := \mathcal{R}(\lambda_n)v := -(v\nabla)\lambda_n + (\nabla v)^t \lambda_n$ for all velocity vectors v.

3. The Newton-Multigrid solver

The KKT-system represents a boundary value problem in the space-time domain. It is shown e.g. in [6] that, assuming sufficient regularity on the state (y, p) and the adjoint state (λ, ξ), it can equivalently be rewritten as higher-order elliptic equation in the space-time domain for either the state or the adjoint state. This indicates that multigrid can be used as efficient solver for the (linearised) KKT system as it is an ideal solver for elliptic PDEs.

We formally define the solution approach as outer nonlinear loop that has to solve a linear subproblem in each nonlinear step.

3.1. The outer defect correction/Newton loop

To treat the nonlinearity in the underlying Navier–Stokes equations, we use a standard nonlinear fixed point iteration as well as a space-time Newton iteration. Both algorithms can be written down as fully discrete preconditioned defect correction loop,

$$\begin{aligned} &1.) \quad C(w_i^h)d_i \;=\; g_i \;:=\; \left(f^h - G^h(w_i^h)w_i^h\right) \\ &2.) \quad w_{i+1}^h \;:=\; w_i^h + d_i. \end{aligned}$$

For the fixed point method, we choose $C(w^h) := G^h(w^h)$ as preconditioner, while the space-time Newton method is characterised by $C(w^h) := F^h(w^h)$ with $F^h(w^h)$ being the discrete analogon to $F(w)$ from Section 2. The solution of the auxiliary problem $C(w_i^h)d_i = g_i$ is obtained by applying the following space-time multigrid method.

3.2. The inner multigrid solver

Let Ω_1, ..., Ω_{NLMAX} be a conforming hierarchy of triangulations of the domain Ω in the sense of [7]. Ω_1 is a basic coarse mesh and Ω_{l+1} stems from a regular refinement of Ω_l (i.e. new vertices, cells and edges are generated by connecting opposite midpoints of edges). We use V_1, ..., V_{NLMAX} to refer to the different Finite Element spaces in space built upon these meshes. Furthermore, let T_1, ..., T_{NLMAX} be a hierarchy of decompositions of the time interval $[0,T]$, where each T_{l+1} stems from T_l by bisecting each time interval. For each l, the above discretisation in space and time yields a solution space $W_l = V_l \times T_l$ and a space-time system

$$G^l w^l = f^l, \qquad l = 1, ..., \mathrm{NLMAX}$$

of the form (9). $f^{\mathrm{NLMAX}} = f^h$, $w^{\mathrm{NLMAX}} = w^h$ and $G^{\mathrm{NLMAX}} = G^h$ identify the discrete right hand side, the solution and system operator on the finest level, respectively.

Algorithm 1 Space-time multigrid

```
function SpaceTimeMultigrid(w;f;l)
    if (l = 1) then
        return (G^1)^{-1} f                                  ▷ coarse grid solver
    end if
    while (not converged) do
        w ← S(G^l, w, f, NSMpre)                             ▷ presmoothing
        d ← R(f − G^l w)                                     ▷ restricion of the defect
        w ← w + P(SpaceTimeMultigrid(0; d; l − 1))           ▷ coarse grid correction
        w ← S(G^l, w, f, NSMpost)                            ▷ postsmoothing
    end while
    return w                                                 ▷ solution
end function
```

Let us denote by $I : W_l \to W_{l+1}$ a prolongation operator and by $R : W_l \to W_{l-1}$ the corresponding restriction. Furthermore, let $S : W_l \to W_l$ define a *smoothing* operator (see the following sections for a definiton of these operators). Let us denote with NSMpre, NSMpost the numbers of pre- and postsmoothing steps, respectively. With these components and definitions, Algorithm 1 implements a basic multigrid V-cycle. For variations of this algorithm which use the W- or F-cycle, see [2, 11, 28]. The algorithm is called on the maximum level by

$$\text{SpaceTimeMultigrid}(w^{\text{NLMAX}};f^{\text{NLMAX}};\text{NLMAX})$$

and implicitely uses the matrices G^1, ..., G^{NLMAX}.

3.3. Prolongation/Restriction

Our discretisation is based on Finite Differences in time and Finite Elements in space. The operators for exchanging solutions and right hand side vectors between the different levels therefore decompose into a time prolongation/restriction and space prolongation/restriction. Let k be the space level, $I_S : V_k \to V_{k+1}$ the prolongation operator in space and $R_S : V_{k+1} \to V_k$ the corresponding restriction. The prolongation for a space-time vector $w^l = (w_0^l, ..., w_N^l)$ on space-time level l can be written as:

$$P(w^l) := \left(P_S(w_0^l),\ \frac{P_S(w_0^l) + P_S(w_1^l)}{2},\ P_S(w_1^l),\ \frac{P_S(w_1^l) + P_S(w_2^l)}{2},\ \dots,\ P_S(w_N^l) \right)$$

and is a composition of the usual Finite Difference prolongation in time (see also [12]) and Finite Element prolongation in space. The corresponding restriction for a defect vector $d^l = (d_0^l, ..., d_{2N}^l)$ follows directly:

$$R(d^l) := \left(R_S(\frac{1}{4}(2d_0^l + d_1^l)),\ R_S(\frac{1}{4}(d_1^l + 2d_2^l + d_3^l)),\ \dots,\ R_S(\frac{1}{4}(d_{2N-1}^l + 2d_{2N}^l)) \right)$$

Our numerical tests in Section 4 are carried out with the nonconforming $\tilde{Q}_1/Q_0$ Finite Element pair in space. For these elements, we use the standard prolongation/restriction operators which can be found e.g. in [18, 22].

3.4. Smoothing operators and coarse grid solver

The special matrix structure of the global space-time matrix (9) allows to define iterative smoothing operators based on defect correction. Note that usually, every smoother can also be used as coarse grid solver to solve the equation $(G^1)^{-1}f$ in the first step of the algorithm; for that purpose, one has to replace the fixed number of iterations by a stopping criterion depending on the residuum.

Let us first introduce some notations. The space-time matrix G^l at level l can be decomposed into the block submatrices as follows with diagonal submatrices

G_i^l for the timesteps $i = 1, ..., N$:

$$G^l = \left(\begin{array}{c|c|c|c} G_0^l & \hat{M}_0^l & & \\ \hline \tilde{M}_1^l & G_1^l & \hat{M}_1^l & \\ \hline & \ddots & \ddots & \ddots \\ \hline & & \tilde{M}_N^l & G_N^l \end{array}\right), \quad G_i^l =: \begin{pmatrix} A_i^{\text{primal}} & M_i^{\text{dual}} & B & 0 \\ M_i^{\text{primal}} & A_i^{\text{dual}} & 0 & B \\ B^T & 0 & 0 & 0 \\ 0 & B^T & 0 & 0 \end{pmatrix}$$

A_i^{primal}, A_i^{dual} are velocity submatrices, M_i^{primal} and M_i^{dual} coupling matrices between the primal and dual velocity and B and B^T clustering the gradient/divergence matrices which are independent of the timestep i. For simplicity, we dropped the index l here. Furthermore, we assume the decompositions $x_i = (x_i^y, x_i^\lambda, x_i^p, x_i^\xi)$ and $d_i = (d_i^y, d_i^\lambda, d_i^p, d_i^\xi)$ of vectors into primal/dual subvectors.

We introduce three iterative block smoothing algorithms. Let $\omega, \omega_1, \omega_2 \in \mathbb{R}$ be damping parameters. The special matrix structure suggests the use of a Block-Jacobi method in the form of Algorithm 2.

Similar to a Block-Jacobi algorithm, it is possible to design a forward-backward block SOR algorithm for smoothing, see Algorithm 3. (For the sake of notation, we define $x_{-1} := x_{N+1} := 0$, $\tilde{M}_0 := \hat{M}_N := 0$.) In contrast to Block-Jacobi, this algorithm exploits basic coupling in time without significant additional costs. The algorithm allows to specify an additional parameter NSMinner which defines how many forward-backward-sweeps are calculated before updating the flow; in our computations however, we always use NSMinner=1.

Above smoothers always treat the primal and dual solution in a coupled way. On the other hand, one can also decouple these solution parts and perform a forward simulation for the primal solution, followed by a backward simulation for the dual solution, see Algorithm 4. This type of algorithm, which we call 'forward-backward simulation algorithm' is rather natural and was used in a similar form by other authors before as a solver (see e.g. [9, 25]). It is expected to be a compromise in speed and stability: Fully coupled systems in space are avoided, so

Algorithm 2 Space-time Block-Jacobi smoother

```
function JacSmoother(G^l,w,f,NSM)
    for j = 0 to NSM do
        d ← f − G^l w                          ▷ Defect
        for i = 0 to N do
            d_i ← (G_i^l)^{-1} d_i             ▷ Block-Jacobi preconditioning
        end for
        w ← w + ωd
    end for
    return w                                   ▷ Solution
end function
```

Algorithm 3 Forward-Backward Block-SOR smoother

function FBSORSmoother(G^l,w,f,NSM)
  for istep = 1 to NSM **do**
    $r \leftarrow f - G^l w$ ▷ Defect
    $x \leftarrow 0$ ▷ correction vector
    for istepinner = 1 to NSMinner **do**
      $x^{\text{old}} \leftarrow x$
      for $i = 0$ to N **do** ▷ Forward in time
        $d_i \leftarrow r_i - \tilde{M}_i x_{i-1} - \hat{M}_i x_{i+1}^{\text{old}}$ ▷ Defect in time
        $x_i \leftarrow (1-\omega_1) x_i^{\text{old}} + \omega_1 (G_i^l)^{-1} d_i$
      end for
      $x^{\text{old}} \leftarrow x$
      for $i = N$ downto 0 **do** ▷ Backward in time
        $d_i \leftarrow r_i - \tilde{M}_i x_{i-1}^{\text{old}} - \hat{M}_i x_{i+1}$ ▷ Defect in time
        $x_i \leftarrow (1-\omega_1) x_i^{\text{old}} + \omega_1 (G_i^l)^{-1} d_i$
      end for
    end for
    $w \leftarrow w + \omega_2 x$ ▷ Correction
  end for
  return w ▷ Solution
end function

the computation of each timestep is faster. On the other hand, due to the reduced coupling, the convergence speed of the overall smoother might be reduced.

Note that the key feature of all algorithms is the solution of saddle point subsystems of the form

$$\begin{pmatrix} A_V & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \quad \Leftrightarrow: \quad A_{\text{sp}} c = d$$

in one time step. A_V contains here all velocity submatrices and B, B^T all gradient/divergence submatrices.. The full space-time algorithm therefore reduces to an algorithm in space. All space-time operations (e.g. matrix-vector multiplications) can therefore be carried out without setting up the whole space time matrix in memory. The system $A_{\text{sp}} c = d$ is a coupled saddle point problem for primal and/or velocity and pressure. It can be solved e.g. by using direct solvers (as long as the number of unknowns in space is not too large) or sophisticated techniques from computational fluid dynamics, namely a spatial multigrid with Pressure-Schur-Complement based smoothers. A typical approach is presented in the next section.

Algorithm 4 Forward-Backward simulation smoother

function FBSimSmoother(G^l,w,f,NSM)
  for istep = 1 to NSM **do**
    $r \leftarrow f - G^l w$ ▷ Defect
    $x \leftarrow 0$ ▷ correction vector
    for $i = 0$ to N **do** ▷ Forward in time
      $d_i \leftarrow r_i - G_i^l x_i - \tilde{M}_i x_{i-1}$
      $\begin{pmatrix} x_i^y \\ x_i^p \end{pmatrix} \leftarrow \begin{pmatrix} x_i^y \\ x_i^p \end{pmatrix} + \omega_1 \begin{pmatrix} A_i^{\text{primal}} & B \\ B^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} d_i^y \\ d_i^p \end{pmatrix}$
    end for
    for $i = N$ downto 0 **do** ▷ Backward in time
      $d_i \leftarrow r_i - G_i^l x_i - \hat{M}_i x_{i+1}$
      $\begin{pmatrix} x_i^\lambda \\ x_i^\xi \end{pmatrix} \leftarrow \begin{pmatrix} x_i^\lambda \\ x_i^\xi \end{pmatrix} + \omega_1 \begin{pmatrix} A_i^{\text{dual}} & B \\ B^T & 0 \end{pmatrix}^{-1} \begin{pmatrix} d_i^\lambda \\ d_i^\xi \end{pmatrix}$
    end for
    $w \leftarrow w + \omega_2 x$ ▷ Correction
  end for
  return w ▷ Solution
end function

3.5. Coupled multigrid solvers in space

Systems of the form $A_{\text{sp}} c = d$ for subproblems in space can efficiently be solved with a multigrid solver in space. For a proper description, we have to formulate prolongation, restriction and smoothing operators. Prolongation and restriction operators based on the applied Finite Element spaces are standard and well known (see e.g. [2, 5, 11, 28, 22]). Smoothing operators acting simultaneously on the primal and dual variables can be constructed e.g. using the pressure Schur complement ('PSC') approach for CFD problems (see also [21, 26, 27]). We shortly describe this approach here; a complete overview and in-depth description of all operators and smoothers will be given in [19].

We first introduce some notations. In each timestep, a linear system $Ax = b$ has to be solved; in our case, this system can be written in the form

$$\begin{pmatrix} A_V & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

which is a typical saddle point problem for primal and/or dual variables, with A_V being a velocity submatrix and B and B^T clustering the gradient/divergence matrices. Depending on the type of the underlying space-time smoother, this system contains either only primal or dual variables, or it represents a combined system of primal and dual variables. In the latter case, A_V, B and B^T decompose into proper 2×2 block submatrices.

Let *iel* denote the number of an arbitrary element in the mesh. Furthermore, let $I(iel)$ identify a list of all degrees of freedom that can be found on element *iel*, containing numbers for the primal and/or dual velocity vectors in all spatial dimensions and the primal and/or dual pressure. With this index set, we define $A_{I(iel)}$ to be a (rectangular) matrix containing only those rows from A identified by the index set $I(iel)$. In the same way, let $x_{I(iel)}$ and $b_{I(iel)}$ define the subvectors of x and b containing only the entries identified by $I(iel)$. Furthermore we define $A_{I(iel),I(iel)}$ to be the (square) matrix that stems from extracting only those rows and columns from A identified by $I(iel)$.

Algorithm 5 PSC-Smoother for smoothing an approximate solution to $Ax = b$

function PSCSmoother(A,x,b,NSM)
  for ism = 1,NSM **do** ▷ NSM smoothing sweeps
    for iel = 1 to NEL **do** ▷ Loop over the elements
      $x_{I(iel)} \leftarrow x_{I(iel)} + \omega C_{iel}^{-1}(b_{I(iel)} - A_{I(iel)}x)$ ▷ Local Correction
    end for
  end for
  return x ▷ Solution
end function

This notation allows to formulate the basic PSC smoother in space, see Algorithm 5; $\omega \in \mathbb{R}$ is used here as a damping parameter with default value $\omega = 1$. Of course, this formulation is not yet complete, as it is lacking a proper definition of the local preconditioner C_{iel}^{-1} which is a small square matrix with as many unknowns as indices in $I(iel)$.

There are two basic approaches for this preconditioner. The first approach, which we entitle by PSCSmootherFull, results in the simple choice of $C_{iel} := A_{I(iel),I(iel)}$ and calculating C_{iel}^{-1} by invoking a LU decomposition, e.g. with the LAPACK package [20]. That approach is rather robust and still feasible as the system is small; for the $\tilde{Q}_1/Q_0$ space that is used in our discretisation (see [22]), the system has 18 unknowns.

The second approach, which we call PSCSmootherDiag, results in taking a different subset of the matrix A for forming $C_{I(iel)}$. To describe this approach, we define

$$\hat{A} := \begin{pmatrix} \mathrm{diag}(A_V) & B \\ B^T & 0 \end{pmatrix}$$

where *diag(·)* refers to the operator taking only the diagonal elements of a given matrix. The local preconditioner can then be formulated as $C_{iel} := \hat{A}_{I(iel),I(iel)}$. If the local system is a combined system of primal and dual variables, this approach decouples the primal from the dual variables. Applying $\hat{A}^{-1}_{I(iel),I(iel)}$ then decomposes into two independent subproblems which is much faster but leads to reduced stability. Most of the numerical tests in the later sections were carried out using PSCSmootherDiag except where noted. We note that it is even possible

to increase the stability by applying this approach to patches of cells (cf. [21]) but we do not apply this approach here.

4. Numerical examples

In this section we numerically analyse the proposed solver strategies with respect to robustness and efficiency. The nonlinearity is captured by a space-time fixed point/Newton iteration, both preconditioned by the proposed space-time multigrid.

4.1. The Driven-Cavity example

Example 4.2 (Driven-Cavity configuration). Let a domain $\Omega = [0,1]^2$ be given. On the four boundary edges $\Gamma_1 := \{0\} \times (0,1)$, $\Gamma_2 := [0,1] \times \{0\}$, $\Gamma_3 := \{1\} \times (0,1)$, $\Gamma_4 := [0,1] \times \{1\}$ we describe Dirichlet boundary conditions as $y(x,t) = (0,0)$ for $x \in \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ and $y(x,t) = (1,0)$ for $x \in \Gamma_4$. The coarse grid consists of only one quadratic element. The time interval for this test case is $[0,T]$ with $T = 1$, the viscosity parameter of the uncontrolled and controlled flow is set to $\nu = 1/400$. The initial flow y^0 is the stationary fully developed Navier–Stokes flow at $\nu = 1/400$, while the target flow z is chosen as the fully developed Stokes–flow.

A stationary analogon of this example was analysed in [1] and in [25] the authors analyse this problems under constraints, see also [13, 14]. Figure 1 shows a picture of the streamlines of the target flow and the initial flow with the corresponding velocity magnitude in the background. For better visualisation, we took a different resolution of the positive and negative streamlines. Figure 2 depicts the controlled flow and the control at $t = 0.075$, $t = 0.25$ and $t = 0.5$. One identifies two main vortices which 'push' the Navier–Stokes flow to the Stokes-Flow at the

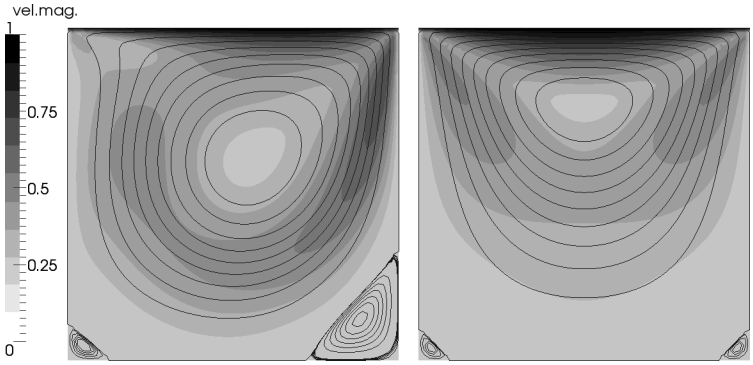

FIGURE 1. *Driven-Cavity* example, Streamlines of the stationary Navier–Stokes (initial) flow (left) and stationary Stokes (target-) flow (right). Velocity magnitude in the background.

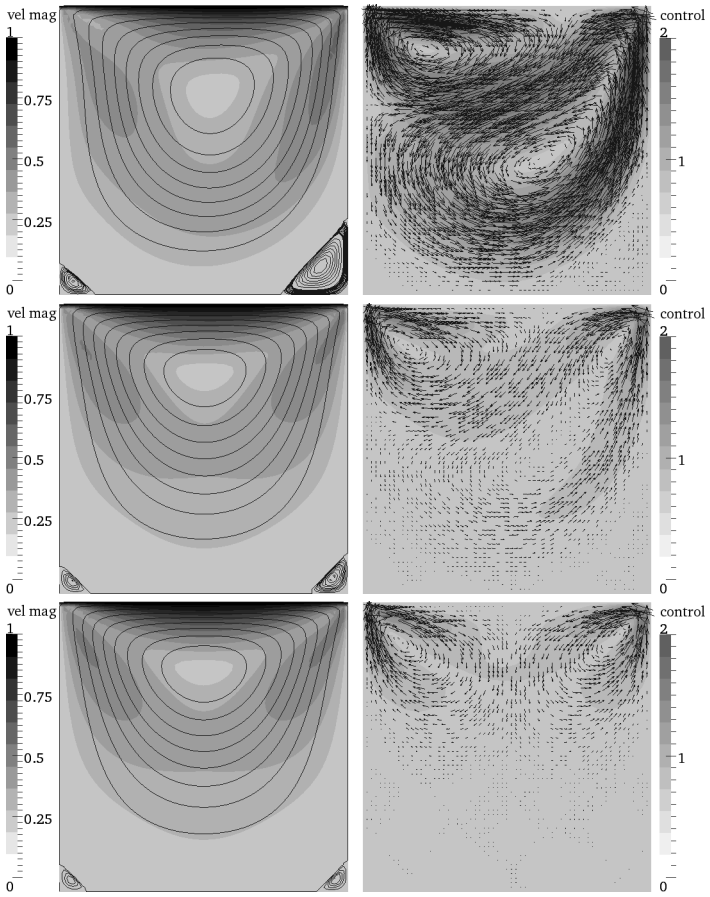

FIGURE 2. *Driven-Cavity* example, controlled flow at $t = 0.075$ (top), $t = 0.25$ (center) and $t = 0.5$ (bottom). Left: Streamlines with primal velocity magnitude in the background. Right: Control.

beginning of the time interval. Table 1 lists the number of unknowns for a pure forward simulation and the optimisation for this problem.

In the first couple of tests we analyse the behaviour of the solver when being applied to the Driven-Cavity example. For our tests, we prototypically choose the regularisation parameters in the KKT-system to be $\alpha = 0.01$ and $\gamma = 0$ and start with defining a basic coarse mesh in space and time. We choose $\Delta t_{\text{coarse}} = 1/10$ and $h_{\text{coarse}} = 1/4$, although any other relation between Δt_{coarse} and h_{coarse} would be suitable as well. This mesh is simultaneously refined by regular refinement in space and time. On each space-time level, we perform the following tests:

		simulation		optimisation	
Δt	h	#DOF space	#DOF total	#DOF space	#DOF total
1/20	1/8	352	7 392	704	14 784
1/40	1/16	1 344	55 104	2 688	110 208
1/80	1/32	5 248	425 088	10 496	850 176
1/160	1/64	20 736	3 338 496	41 472	6 676 992

TABLE 1. *Driven-Cavity* example, problem size. Number of degrees of freedom in space ('#DOF space') and on the whole space-time domain including the initial condition ('#DOF total').

1.) We calculate an optimal control problem with the target flow as specified above. The nonlinear space-time solver damps the norm of the residual by $\varepsilon_{\mathrm{OptNL}} = 10^{-5}$, the linear space-time multigrid in each nonlinear iteration by $\varepsilon_{\mathrm{OptMG}}$. The convergence criterion of the innermost spatial multigrid solver in each timestep was set to damp the norm of the residual by $\varepsilon_{\mathrm{SpaceMG}}$.

2.) We calculate a pure simulation with a fully implicit Navier–Stokes solver in time, using the control computed in 1.) as right hand side. In each timestep the norm of the residual was damped by $\varepsilon_{\mathrm{SimNL}} = 10^{-5}$. The linear multigrid subsolver in each nonlinear iteration damps the norm of the residual by $\varepsilon_{\mathrm{SimMG}}$.

General tests. In the first couple of tests we analyse the behaviour of the nonlinear space-time solver for optimal control. We fix the space-time mesh to $\Delta t = 1/40$ and $h = 1/16$ which is already fine enough for a qualitative analysis. The convergence criterion of the innermost solver to $\varepsilon_{\mathrm{SpaceMG}} = 10^{-2}$. The smoother in space is PSCSMOOTHERDIAG, the space-time smoother FBSORSMOOTHER($\omega_1 = 0.8$, $\omega_2 = 1$). From Figure 3 one can see linear convergence for the fixed point iteration and quadratic convergence of the Newton iteration. Note that because of $\varepsilon_{\mathrm{OptNL}} = 10^{-5}$ the impact of the parameter $\varepsilon_{\mathrm{OptMG}}$ to Newton is rather low, it does not (yet) influence the number of iterations.

The next test analyses the influence of the innermost stopping criterion $\varepsilon_{\mathrm{SpaceMG}}$. For the space-time smoothers JACSMOOTHER($\omega = 0.7$, NSM $= 4$), FBSIMSMOOTHER($\omega_1 = 0.8, \omega_2 = 0.5$, NSM $= 4$) and FBSORSMOOTHER($\omega_1 = 0.8, \omega_2 = 1$, NSM $= 1$) we fix the convergence criterion of the space-time multigrid to $\varepsilon_{\mathrm{OptMG}} = 10^{-2}$ (Table 2). Then, we calculate the fixed point and Newton iteration for different settings of $\varepsilon_{\mathrm{SpaceMG}}$. As one can see from the tables, the solver behaves very robust against $\varepsilon_{\mathrm{SpaceMG}}$, so we choose $\varepsilon_{\mathrm{SpaceMG}} = 10^{-2}$ for all later tests. Furthermore one can see that the efficiency of FBSIMSMOOTHER with four smoothing steps is comparable to FBSORSMOOTHER with one smoothing step. JACSMOOTHER on the other hand is the least efficient smoother of these three, so we omit further investigations of it.

Table 3 reveals that for a reasonable convergence criterion of $\varepsilon_{\mathrm{OptNL}} = 10^{-5}$, the number of linear and nonlinear iterations is rather independent of the convergence criterion $\varepsilon_{\mathrm{OptMG}}$ of the space-time multigrid solver. (The smoother in these

computations is FBSORSmoother($\omega_1 = 0.8, \omega_2 = 1$, NSM $= 1$).) Furthermore, the number of linear and nonlinear iterations stays almost constant upon increasing the resolution of the space-time mesh which confirms the linear complexity of the algorithm. More precisely, the number of iterations even reduces with increasing space-time level – an effect which was also observed and proven in [6].

We note that the impact of the setting of $\varepsilon_{\text{OptMG}}$ would be stronger if $\varepsilon_{\text{OptNL}}$ is set smaller; for such cases, one would prefer an inexact Newton algorithm which adaptively determins the stopping criterion of the linear space-time solver; for details see e.g. [14]. In all further tests, we take $\varepsilon_{\text{OptMG}} = 10^{-2}$.

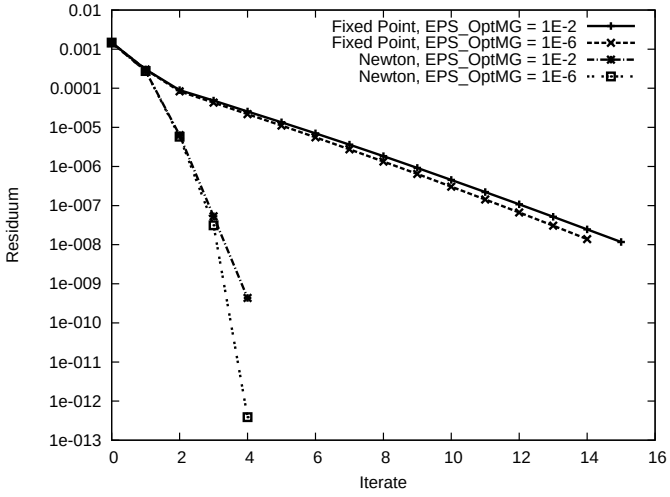

FIGURE 3. *Driven-Cavity* example, convergence of the fixed point and Newton algorithm.

		JACSmoother			FBSimSmoother			FBSORSmoother		
Slv.	$\varepsilon_{\text{SpMG}}$	#NL	#MG	Time	#NL	#MG	Time	#NL	#MG	Time
FP	10^{-1}	15	134	811	15	60	350	15	75	310
	10^{-2}	15	134	846	15	60	352	15	75	319
	10^{-6}	15	134	2405	15	60	650	15	75	775
N	10^{-1}	4	50	449	4	16	138	4	25	150
	10^{-2}	4	50	467	4	16	138	4	25	160
	10^{-6}	4	50	1181	4	16	251	4	25	349

TABLE 2. The influence of $\varepsilon_{\text{SpaceMG}}$ to the Fixed point ('FP') and Newton ('N') solver. $\varepsilon_{\text{OptMG}} = 10^{-2}$. '#NL'=number of iterations, nonlinear solver. '#MG'=number of iterations, space-time multigrid. 'Time'= comp. time in sec.; *Driven-Cavity* example.

$\varepsilon_{\text{OptMG}}$:		10^{-2}				10^{-6}			
Nonl. Solv.:		fixed point		Newton		fixed point		Newton	
Δt	h	#NL	#MG	#NL	#MG	#NL	#MG	#NL	#MG
1/40	1/16	15	75	4	25	14	252	4	80
1/80	1/32	8	40	4	25	8	158	4	87
1/160	1/64	6	33	4	27	6	132	3	68

TABLE 3. The influence of $\varepsilon_{\text{OptMG}}$. '#NL'=number of iterations, nonlinear solver. '#MG'=number of iterations, space-time multigrid. *Driven-Cavity* example.

Optimisation vs. Simulation. In the following tests we compare the behaviour of the solver for the optimal control problem with a pure simulation. As convergence criterion for the solver we choose $\varepsilon_{\text{SimNL}} = \varepsilon_{\text{OptNL}} = 10^{-5}$ and $\varepsilon_{\text{SimMG}} = \varepsilon_{\text{OptMG}} = 10^{-2}$. Table 4 depicts the result of a set of forward simulations for various settings of Δh and t. We tested both nonlinear solvers, the simple nonlinear fixed point iteration (entitled by 'FP') as well as the Newton iteration (entitled by 'N'). The linear subproblems in space were solved by a multigrid solver in space with a PSCSMOOTHERDIAG-type smoother, the space-time multigrid used an FBSORSMOOTHER($\omega_1 = 0.8, \omega_2 = 1$, NSM $= 1$)-smoother.

The columns ⌀#NL and ⌀#MG in this table describe the average number of linear/nonlinear iterations per timestep in the simulation, which is comparable to the number of nonlinear/linear iterations of the optimisation (columns #NL and #MG, compare also Table 3). #MG in the optimisation task differs from ⌀#MG in the simulation by a factor of approx. 2–3, independent of the level, which means that the effort for both, the simulation and the optimisation, grows with same complexity when increasing the problem size.

			simulation		optimisation				
Slv.	Δt	h	⌀#NL	⌀#MG	#NL	#MG	T_{sim}	T_{opt}	$\frac{T_{\text{opt}}}{T_{\text{sim}}}$
FP	1/40	1/16	4	15	15	75	3	316	124
	1/80	1/32	4	16	8	40	21	1414	68
	1/160	1/64	3	16	6	33	207	10436	51
N	1/40	1/16	3	9	4	25	3	158	63
	1/80	1/32	3	10	4	25	21	1359	63
	1/160	1/64	2	11	4	27	219	11359	52
N	1/16	1/16	3	12	4	13	1.2	46	38
	1/32	1/32	3	13	4	13	10.4	330	32
	1/64	1/64	3	13	4	12	109.9	2275	21

TABLE 4. *Driven-Cavity*-example, optimisation and simulation. Execution time as well as mean number of iterations for a pure forward simulation and the corresponding optimisation.

	coarse mesh		fine mesh		FBSORSmoother			FBSimSmoother		
Slv.	Δt	h	Δt	h	#NL	#MG	T_{opt}	#NL	#MG	T_{opt}
FP	1/4	1/4	1/16	1/16	15	45	99	div	div	div
	1/6	1/4	1/24	1/16	15	59	164	15	46	176
	1/8	1/4	1/32	1/16	15	60	197	15	59	279
N	1/4	1/4	1/16	1/16	4	13	45	div	div	div
	1/6	1/4	1/24	1/16	4	17	74	div	div	div
	1/8	1/4	1/32	1/16	4	21	117	4	15	109

TABLE 5. *Driven-Cavity*-example. Smoother robustness.

Table 4 furthermore compares the different execution times of the simulation and optimisation solver. Using the Newton method clearly shows that the execution time of the optimisation is a bounded multiple of the execution time of the simulation. One can see a factor of $C \approx 50 - 60$ for this example[1], even being better for higher mesh resolutions. This factor depends strongly on the anisotropy of the space-time coarse mesh: The lower part of the table shows the results for a time coarse mesh with $\Delta t_{\text{coarse}} = 1/4$ instead of $\Delta t_{\text{coarse}} = 1/10$, which is fully isotropic in space and time. For this mesh, we obtain factors of $C \approx 20 - 30$.

Table 5 compares the FBSIMSMOOTHER against the FBSORSMOOTHER in terms of robustness. Both algorithms have a similar efficiency, but the solver does not converge anymore for large Δt_{coarse} if FBSIMSMOOTHER is used. Furthermore one can see, that the number of space-time multigrid steps #MG (and thus the solver time T_{opt}) increases with finer Δt_{coarse}. This behaviour is typical for SOR-like algorithms and depends on the anisotropy in the space-time mesh.

4.3. Tests with the Flow-around-Cylinder example

In a similar way as above, we now carry out a set of tests for the more complicated *Flow-around-Cylinder* problem, which is a modification of a well known CFD benchmark problem in [22]:

Example 4.4 (Flow-around-Cylinder configuration). As spatial domain, we prescribe a rectangle without an inner cylinder $\Omega := [0, 2.2] \times [0, 0.41] \setminus B_r(0.2, 0.2)$, $r = 0.05$. We decompose the boundary of this domain into five parts: $\Gamma_1 := \{0\} \times [0, 0.41]$, $\Gamma_2 := (0, 2.2] \times \{0\}$, $\Gamma_3 := \{2.2\} \times (0, 0.41)$, $\Gamma_4 := [0, 2.2] \times \{0.41\}$ and $\Gamma_5 := \partial B_r(0.2, 0.2)$. Boundary conditions are: $y(x, t) := (0, 0)$ for $x \in \Gamma_2 \cup \Gamma_4 \cup \Gamma_5$, do-nothing boundary conditions on Γ_3 and a parabolic inflow profile with maximum velocity $U_{\max} := 0.3$ on Γ_1. The time interval for this test case is $[0, T]$ with $T = 1$. Similar to the previous *Driven-Cavity*-example, our initial flow is the

[1] We note here that the large factors for 'small' mesh resolutions ($h \leq 1/16$) are merely an effect of acceleration due to cache effects on the side of the simulation and can be expected: A simulation with only some hundreds of unknowns is able to rather completely run in the cache, whereas the optimisation of a fully nonstationary PDE in space and time usually does not fit into the computer cache anymore. A more detailed analysis and exploitation of the computer cache and its effects can be found in [23, 24].

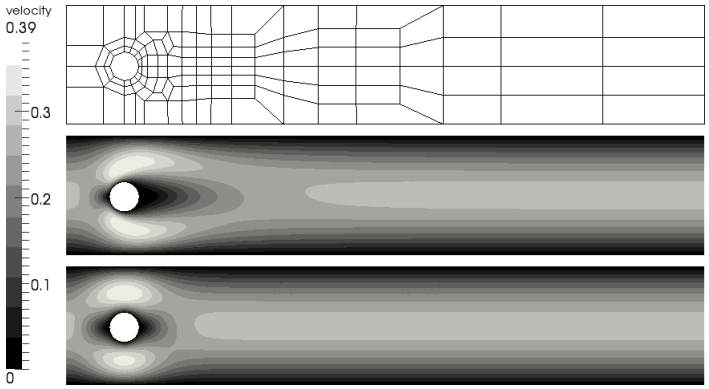

FIGURE 4. *Flow-around-Cylinder* example. Velocity magnitude. Top: Coarse mesh. Center: Initial Navier–Stokes flow. Bottom: Target Stokes flow.

stationary Navier–Stokes-flow at $\nu = 1/500$ while the target flow is the stationary Stokes-flow. The viscosity parameter in the optimisation is set to $\nu = 1/500$ as well (resulting in a Re=10 optimisation).

Figure 4 depicts the basic mesh, the initial flow and the target flow. For the time discretisation, we choose $N = 10$ timesteps on time level 1 which leads to 20, 40 and 80 timesteps on space-time level 2, 3 and 4, resp. In Table 6 the corresponding problem size for the simulation and optimisation can be seen. The convergence criteria for the solvers are defined as $\varepsilon_{\mathrm{SimNL}} = \varepsilon_{\mathrm{OptNL}} = 10^{-5}$ and $\varepsilon_{\mathrm{SimMG}} = \varepsilon_{\mathrm{OptMG}} = \varepsilon_{\mathrm{SpaceMG}} = 10^{-2}$. We again focus on the difference in the execution time between the simulation and a corresponding optimisation and proceed as in the last section: We first compute a control with the optimisation and then execute a pure simulation with the computed control as right hand side.

Table 7 compares the mean number of nonlinear and linear iterations per timestep in the simulation to the number of nonlinear and linear iterations in the optimisation. We use FBSIMSMOOTHER($\omega_1 = 0.8$, $\omega_2 = 0.5, \mathrm{NSM} = 4$) here as space-time smoother. Like in the Driven-Cavity example, the number of nonlinear iterations for the optimisation is comparable to the mean number of nonlinear iterations in the simulation, and so it does for the number of linear iterations. The execution time of the simulation and the optimisation[2] differs by only a constant factor which is typically decreasing for higher levels of refinement. The table indicates a factor $C \approx 20 - 30$ for reasonable levels.

[2]In the table, the execution time for the simulation using the fixed point algorithm is usually lower than the execution time with the Newton algorithm. This stems from the fact that when using Newton, the effort for solving the spatial system in every timestep is much higher, and due to the convergence criterion $\varepsilon_{\mathrm{OptNL}} < 10^{-5}$ the Newton and the fixed point method need approximately the same number of linear/nonlinear iterations. This situation usually changes if a higher Re number is used as this implies a stronger influence of the nonlinearity!

		simulation		optimisation	
Δt	Space-Lv.	#DOF space	#DOF total	#DOF space	#DOF total
1/20	2	2 704	29 744	5 408	59 488
1/40	3	10 608	222 768	21 216	445 536
1/80	4	42 016	1 722 656	84 032	3 445 312

TABLE 6. *Flow-around-Cylinder* example, problem size. Number of degrees of freedom in space ('#DOF space') and on the whole space-time domain including the initial condition ('#DOF total').

			Simulation		Optimisation				
Slv.	Δt	Space-Lvl.	⌀#NL	⌀#MG	#NL	#MG	T_{sim}	T_{opt}	$\frac{T_{\text{opt}}}{T_{\text{sim}}}$
FP	1/20	2	4	19	4	14	3	111	37
	1/40	3	4	19	4	17	37	972	26
	1/80	4	3	20	4	19	324	9383	29
N	1/20	2	3	12	3	10	3	112	37
	1/40	3	3	15	3	13	61	924	15
	1/80	4	3	14	3	14	387	9410	24

TABLE 7. *Flow-around-Cylinder* example. Simulation and optimisation.

5. Conclusions

Optimal control of the time-dependent Navier–Stokes equations can be carried out with iterative nonlinear and linear solution methods that act on the whole space-time domain. As nonlinear solver Newton's method is used. Because of the special structure of the system matrix a space-time multigrid algorithm can be formulated for the linear subproblems in the Newton iteration. All matrix vector multiplications and smoothing operations can be reduced to local operations in space, thus avoiding the necessity of storing the whole space-time matrix in memory. Problems in space can be tackled by efficient spatial multigrid and Pressure-Schur-Complement techniques from Computational Fluid Dynamics. The overall solver works with optimal complexity, the numerical effort growing linearly with the problem size. The execution time necessary for the optimisation is a bounded multiple of the execution time necessary for a 'similar' simulation, where numerical tests indicate a factor of $C \approx 20-30$ for reasonable configurations. Being based on finite elements, the solver can be applied to rather general computational meshes.

This article concentrated on the basic ingredients of the solver. For simplicity, we restricted to first order implicit Euler discretisation in time and ignored any restrictions on the controls. Nevertheless, higher order schemes like Crank-Nicolson are possible and preferrable for larger timesteps, but the additional coupling leads to a more complicated matrix structure. Similarly, bounds on the control, other finite element spaces for higher accuracy and stabilisation techniques which are necessary to compute with higher Reynolds numbers are topics which have to be addressed in the future.

References

[1] G. V. Alekseyev and V. V. Malikin. Numerical analysis of optimal boundary control problems for the stationary navier-stokes equations. *Computational Fluid Dynamics Journal*, 3(1):1–26, 1994.

[2] R. E. Bank and T. F. Dupond. An optimal order process for solving finite element equations. *Math. Comput.*, 36(153):35–51, 1981.

[3] G. Bärwolff and M. Hinze. Optimization of semiconductor melts. *Zeitschrift für Angewandte Mathematik und Mechanik*, 86:423–437, 2006.

[4] A. Borzi. Multigrid methods for parabolic distributed optimal control problems. *Journal of Computational and Applied Mathematics*, 157:365–382, 2003.

[5] S. C. Brenner. An optimal-order multigrid method for P_1 nonconforming finite elements. *Math. Comput.*, 52(185):1–15, 1989.

[6] G. Büttner. *Ein Mehrgitterverfahren zur optimalen Steuerung parabolischer Probleme*. PhD thesis, Fakultät II – Mathematik und Naturwissenschaften der Technischen Universität Berlin, 2004. `http://edocs.tu-berlin.de/diss/2004/buettner_guido.pdf`.

[7] Ph. G. Ciarlet. *The finite element method for elliptic problems*. Studies in mathematics and its applications, Vol. 4. North-Holland Publishing Company, Amsterdam, New-York, Oxford, 1978. ISBN 0444850287.

[8] H. Goldberg and F. Tröltzsch. On a SQP–multigrid technique for nonlinear parabolic boundary control problems. In W. W. Hager and P. M. Pardalos, editors, *Optimal Control: Theory, Algorithms, and Applications*, pages 154–174. Kluwer, 1998.

[9] M. Gunzburger, E. Ozugurlu, J. Turner, and H. Zhang. Controlling transport phenomena in the czochralski crystal growth process. *Journal of Crystal Growth*, 234:47–62, 2002.

[10] W. Hackbusch. Fast solution of elliptic optimal control problems. *J. Opt. Theory and Appl.*, 31(4):565–581, 1980.

[11] W. Hackbusch. *Multi-Grid Methods and Applications*. Springer, Berlin, 1985. ISBN 3-540-12761-5.

[12] W. Hackbusch. Multigrid methods for FEM and BEM applications. In E. Stein, R. de Borst, and Th. J. R. Hughes, editors, *Encyclopedia of Computational Mechanics*, chapter 20. John Wiley & Sons Ltd., 2004.

[13] M. Hintermüller and M. Hinze. A SQP-semi-smooth Newton-type algorithm applied to control of the instationary Navier–Stokes system subject to control constraints. *Siam J. Optim.*, 16:1177–1200, 2006.

[14] M. Hinze. Optimal and instantaneous control of the instationary navier–stokes equations. Institut für Numerische Mathematik, Technische Universität Dresden, 2000. Habilitation.

[15] M. Hinze and S. Ziegenbalg. Optimal control of the free boundary in a two-phase stefan problem. *J. Comput. Phys.*, 223:657–684, 2007.

[16] M. Hinze and S. Ziegenbalg. Optimal control of the free boundary in a two-phase stefan problem with flow driven by convection. *Z. Angew. Math. Mech.*, 87:430–448, 2007.

[17] M. Hinze and S. Ziegenbalg. Optimal control of the phase interface during solidification of a GaAs melt. *Proc. Appl. Math. Mech.*, 311(8):2501–2507, 2008.

[18] M. Köster. Robuste Mehrgitter-Krylowraum-Techniken für FEM-Verfahren, 2007. Diplomarbeit, Universität Dortmund, Diplomarbeit, `http://www.mathematik.tu-dortmund.de/lsiii/static/schriften_eng.html`.

[19] M. Köster. *A Hierarchical Flow Solver for Optimisation with PDE Constraints*. PhD thesis, TU Dortmund, To appear 2010.

[20] NETLIB. LAPACK – Linear Algebra PACKage, 1992. `http://www.netlib.org/lapack/`.

[21] R. Schmachtel. *Robuste lineare und nichtlineare Lösungsverfahren für die inkompressiblen Navier–Stokes-Gleichungen*. PhD thesis, TU Dortmund, June 2003. `http://www.mathematik.tu-dortmund.de/lsiii/static/schriften_eng.html`.

[22] S. Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. Springer, Berlin, 1999. ISBN 3-540-65433-X.

[23] S. Turek, Ch. Becker, and S. Kilian. Hardware–oriented numerics and concepts for PDE software. *Future Generation Computer Systems*, 22(1–2):217–238, 2006. doi: 10.1016/j.future.2003.09.007.

[24] S. Turek, D. Göddeke, Ch. Becker, S. H. M. Buijssen, and H. Wobker. FEAST – realisation of hardware-oriented numerics for HPC simulations with finite elements. *Concurrency and Computation: Practice and Experience*, 2010. Special Issue Proceedings of ISC 2008, accepted.

[25] M. Ulbrich. Constrained optimal control of Navier–Stokes flow by semismooth newton methods. *Systems Control Lett.*, 48:297–311, 2003.

[26] S. P. Vanka. Block-implicit multigrid solution of Navier-Stokes equations in primitive variables. *Journal of Computational Physics*, 65:138–158, 1986.

[27] H. Wobker and S. Turek. Numerical studies of Vanka-type smoothers in computational solid mechanics. *Advances in Applied Mathematics and Mechanics*, 1(1):29–55, 2009.

[28] H. Yserentant. Old and new convergence proofs for multigrid methods. *Acta Numerica*, pages 1–44, 1992.

Michael Hinze
Department of Mathematics, University of Hamburg, Bundesstrasse 55, 20146 Hamburg, Germany
e-mail: `michael.hinze@uni-hamburg.de`

Michael Köster
Corresponding author. Institut für Angewandte Mathematik, Technische Universität Dortmund, Vogelpothsweg 87, D-44227 Dortmund, Germany
e-mail: `michael.koester@mathematik.tu-dortmund.de`

Stefan Turek
Institut für Angewandte Mathematik, Technische Universität Dortmund, Vogelpothsweg 87, D-44227 Dortmund, Germany
e-mail: `stefan.turek@mathematik.tu-dortmund.de`