Hamburger Beiträge
zur Angewandten Mathematik

A space-time multigrid solver for distributed control of the time-dependent Navier-Stokes system

Michael Hinze, Michael Köster, Stefan Turek

Nr. 2008-12
Dezember 2008

A space-time multigrid solver for distributed control of the time-dependent Navier-Stokes system

Michael Hinze * Michael Köster[†‡] Stefan Turek [§]

December 15, 2008

Abstract

We present a space-time 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. 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 number of unknowns of the discrete problem and robust with regard to the considered flow configuration. A set of numerical examples analyses the solver behaviour for various problem settings with respect to the efficiency of this approach.

1 Introduction

In a recent publication [11] we presented a method to solve a fully nonstationary optimal control problem based on the Stokes equation with a space-time multigrid solver. In this paper we show, that it is possible to extend this concept to a the active control of the fully nonstationary Navier–Stokes equation. Such problems appear in many practical applications like in crystal growth processes [7, 12, 13], where the flow in the melt has a significant impact on the quality of the crystal.

The underlying mathematical formulation is a minimisation problem with PDE constraints. Its first order necessary optimality conditions, the so called Karush-Kuhn-Tucker (KKT)-system, couples the state equation of the PDE to be controlled by an adjoint and a dual equation for the control input. By exploiting the special structure of the system, we were able in [11] to develop a hierarchical solution approach for the optimisation of the Stokes equation which satisfies

$$\frac{\text{effort for optimisation}}{\text{effort for simulation}} \leq C, \tag{1.1}$$

*Department of Mathematics, University of Hamburg, Bundesstrasse 55, 20146 Hamburg, Germany, michael.hinze@uni-hamburg.de

†corresponding author

‡Institute of Applied Mathematics, Technische Universität Dortmund, Vogelpothsweg 87, D-44227 Dortmund, Germany, michael.koester@mathematik.tu-dortmund.de

§Institute of Applied Mathematics, Technische Universität Dortmund, Vogelpothsweg 87, D-44227 Dortmund, Germany, stefan.turek@mathematik.tu-dortmund.de

for a constant $C > 0$ of moderate size. Numerical tests showed a factor of $C \approx 8 - 11$, independent of the refinement level. Here, the effort for the simulation is assumed to be optimal in that sense, that a solver needs about $O(N)$ operations, $N \in \mathbb{N}$ 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 (1.1), the developed solution approach has also complexity $O(N)$. This is in contrast to the adjoint-based steepest descent methods used to solve optimisation problems in many practical applications, which in general do not satisfy this complexity requirement.

In the present work, we propose an extension to this approach for the distributed control of time-dependent Navier–Stokes flow with $O(N)$ complexity. It is based on a space-time Newton method combined with a space-time multigrid approach. This combination is applied to the space-time boundary value problem stemming from the KKT system. First numerical results indicate that solving the KKT-system with this approach is about $C \approx 10 - 30$ times more expensive than the simulation. A related approach can be found, e.g. in [3] 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 [5] a space-time multigrid method for Hackbusch's *integral equation approach* [8] is developed, compare also [6].

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

$$J(y,u) := \frac{1}{2}||y-z||^2_{L^2(Q)} + \frac{\alpha}{2}||u||^2_{L^2(Q)} + \frac{\gamma}{2}||y(T)-z(T)||^2_{L^2(\Omega)} \quad \longrightarrow \quad \text{min!} \tag{2.1}$$

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

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.

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

$$
\begin{aligned}
y_t - \nu\Delta y + y\nabla y + \nabla p &= -\frac{1}{\alpha}\lambda, \qquad (2.2)\\
-\text{div } y &= 0,\\
y(0,\cdot) &= y_0,
\end{aligned}
$$

$$
\begin{aligned}
-\lambda_t - \nu\Delta\lambda - y\nabla\lambda + (\nabla y)^t\lambda + \nabla\xi &= y - z, \qquad (2.3)\\
-\text{div } \lambda &= 0,\\
\lambda(T) &= \gamma(y(T) - z(T)).
\end{aligned}
$$

where we call (2.2) the *primal* and (2.3) the *dual* equation.

Coupled discretisation in time

In the next step, we semi-discretise in time. For stability reasons (cf. [18]) we prefer implicit time stepping techniques that allow a large timestep and therefore lead to less unknowns in time. For the sake of simplicity, we restrict to the standard 1st order backward Euler scheme as a representative of implicit schemes. Schemes of higher order and non-equidistant time stepping will be investigated in a forthcoming paper. For the time discretisation of (2.2) this yields

$$
\begin{aligned}
\frac{y_{n+1} - y_n}{\Delta t} - \nu\Delta y_{n+1} + y_{n+1}\nabla y_{n+1} + \nabla p_{n+1} &= -\frac{1}{\alpha}\lambda_{n+1} \qquad (2.4)\\
-\text{div } y_{n+1} &= 0
\end{aligned}
$$

where $N\in\mathbb{N}$, $n = 0, ..., N-1$ and $\Delta t = 1/N$. To (2.2), (2.3) we apply the discretisation recipe from [2]. For this purpose, we define the following operators: $\mathcal{A}v := -\nu\Delta v$, $\mathcal{I}v := v$, $\mathcal{G}q := \nabla q$, $\mathcal{D}v := -\text{div } v$, $\mathcal{K}_i v := \mathcal{K}(y_i)v := (y_i\nabla)v$, $\overline{\mathcal{K}}_i v := \overline{\mathcal{K}}(y_i)v := (v\nabla)y_i$ and $\mathcal{C}_i v := \mathcal{C}(y_i)v := \mathcal{A}v + \mathcal{K}(y_i)v$ for all velocity vectors v and and pressure functions q in space.

With $x := (y_0, p_0, y_1, p_1, ..., y_n, p_n)$, this yields the nonlinear system of the primal equation,

$$\mathcal{H}x := \mathcal{H}(x)x = \left(\begin{array}{c|c|c|c} \frac{\mathcal{I}}{\Delta t} + \mathcal{C}_0 \quad \mathcal{G} & & & \\ \mathcal{D} & & & \\ \hline -\frac{\mathcal{I}}{\Delta t} & \frac{\mathcal{I}}{\Delta t} + \mathcal{C}_1 \quad \mathcal{G} & & \\ & \mathcal{D} & & \\ \hline & \ddots & \ddots \quad \ddots & \\ \hline & & -\frac{\mathcal{I}}{\Delta t} & \frac{\mathcal{I}}{\Delta t} + \mathcal{C}_n \quad \mathcal{G} \\ & & & \mathcal{D} \end{array}\right) \left(\begin{array}{c} y_0 \\ p_0 \\ y_1 \\ p_1 \\ \vdots \\ y_n \\ p_n \end{array}\right) = \left(\begin{array}{c} (\frac{\mathcal{I}}{\Delta t} + \mathcal{C}_0)y^0 \\ 0 \\ \hline -\frac{\lambda_1}{\alpha} \\ 0 \\ \hline \vdots \\ \hline -\frac{\lambda_n}{\alpha} \\ 0 \end{array}\right), \quad (2.5)$$

which is equivalent to (2.4) if y^0 is solenoidal. In the second step, we focus on the Fréchet derivative of the Navier–Stokes equation. 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

$$\mathcal{M}\bar{x} := \mathcal{M}(x)\bar{x} = \left(\begin{array}{c|c|c|c} \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_0 \quad \mathcal{G} & & & \\ \mathcal{D} & & & \\ \hline -\frac{\mathcal{I}}{\Delta t} & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_1 \quad \mathcal{G} & & \\ & \mathcal{D} & & \\ \hline & \ddots & \ddots \quad \ddots & \\ \hline & & -\frac{\mathcal{I}}{\Delta t} & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_n \quad \mathcal{G} \\ & & & \mathcal{D} \end{array}\right) \left(\begin{array}{c} \bar{y}_0 \\ \bar{p}_0 \\ \bar{y}_1 \\ \bar{p}_1 \\ \vdots \\ \bar{y}_n \\ \bar{p}_n \end{array}\right) \quad (2.6)$$

with the additional operator $\mathcal{N}_i := \mathcal{N}(y_i) := \mathcal{A} + \mathcal{K}(y_i) + \overline{\mathcal{K}}(y_i)$. 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 $\mathcal{N}_i^* := \mathcal{N}^*(y_i) = \mathcal{A} - \mathcal{K}(y_i) + \overline{\mathcal{K}}^*(y_i)$, $\overline{\mathcal{K}}^*(y_i)v = (\nabla y)^t v$ for all velocity vectors v, this reads

$$\mathcal{M}^*\lambda = \mathcal{M}^*(x)\lambda = \left(\begin{array}{c|c|c|c} \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_0^* \quad \mathcal{G} & -\frac{\mathcal{I}}{\Delta t} & & \\ \mathcal{D} & & & \\ \hline & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_1^* \quad \mathcal{G} & -\frac{\mathcal{I}}{\Delta t} & \\ & \mathcal{D} & & \\ \hline & & \ddots \quad \ddots & \ddots \\ \hline & & & \frac{\mathcal{I}}{\Delta t} + \mathcal{N}_n^* \quad \mathcal{G} \\ & & & \mathcal{D} \end{array}\right) \left(\begin{array}{c} \lambda_0 \\ \xi_0 \\ \lambda_1 \\ \xi_1 \\ \vdots \\ \lambda_n \\ \xi_n \end{array}\right) = \left(\begin{array}{c} 0 \\ 0 \\ \hline y_1 - z_1 \\ 0 \\ \hline \vdots \\ \hline (1 + \frac{\gamma}{\Delta t})(y_n - z_n) \\ 0 \end{array}\right) \quad (2.7)$$

which corresponds to the time discretisation scheme

$$\begin{aligned} \frac{\lambda_n - \lambda_{n+1}}{\Delta t} - \nu\Delta\lambda_n - y_n\nabla\lambda_n + (\nabla y_n)^t\lambda_n + \nabla\xi_n &= y_n - z_n \\ -\text{div } \lambda_n &= 0. \end{aligned} \tag{2.8}$$

of (2.3). Here we have used $\mathcal{D}^* = \mathcal{G}$ and $\mathcal{A}^* = \mathcal{A}$. Now let us define $w_i := (y_i, p_i, \lambda_i, \xi_i)$ and $w := (w_0, w_1, ...) := (y_0, p_0, \lambda_0, \xi_0, y_1, p_1, \lambda_1, \xi_1, y_2, p_2, \lambda_2, \xi_2, ...)$. After shifting the terms with λ_{n+1} and y_n in (2.4) and (2.8) 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

$$G(w)w = f. \tag{2.9}$$

Here,

$G = G(w) =$

$$\left(\begin{array}{cccc|cccc|cccc|cccc} \frac{\mathcal{I}}{\Delta t}+\mathcal{C}_0 & \mathcal{G} & 0 & 0 & 0 & & & & & & & & & & & \\ \mathcal{D} & 0 & 0 & 0 & & 0 & & & & & & & & & & \\ -\mathcal{I} & 0 & \frac{\mathcal{I}}{\Delta t}+\mathcal{N}_0^* & \mathcal{G} & & & -\frac{\mathcal{I}}{\Delta t} & & & & & & & & & \\ 0 & 0 & \mathcal{D} & 0 & & & & 0 & & & & & & & & \\ \hline -\frac{\mathcal{I}}{\Delta t} & & & & \frac{\mathcal{I}}{\Delta t}+\mathcal{C}_1 & \mathcal{G} & \frac{\mathcal{I}}{\alpha} & 0 & 0 & & & & & & & \\ & 0 & & & \mathcal{D} & 0 & 0 & 0 & & 0 & & & & & & \\ & & 0 & & -\mathcal{I} & 0 & \frac{\mathcal{I}}{\Delta t}+\mathcal{N}_1^* & \mathcal{G} & & & -\frac{\mathcal{I}}{\Delta t} & & & & & \\ & & & 0 & 0 & 0 & \mathcal{D} & 0 & & & & 0 & & & & \\ \hline & & & & \ddots & & & & \ddots & & & & \ddots & & & \\ \hline & & & & & & & & -\frac{\mathcal{I}}{\Delta t} & & & & \frac{\mathcal{I}}{\Delta t}+\mathcal{C}_n & \mathcal{G} & \frac{\mathcal{I}}{\alpha} & 0 \\ & & & & & & & & & 0 & & & \mathcal{D} & 0 & 0 & 0 \\ & & & & & & & & & & 0 & & -(1+\frac{\gamma}{\Delta t})\mathcal{I} & 0 & \frac{\mathcal{I}}{\Delta t}+\mathcal{N}_n^* & \mathcal{G} \\ & & & & & & & & & & & 0 & 0 & 0 & \mathcal{D} & 0 \end{array}\right)$$

and the right hand side is given by

$$f = \Big(\underbrace{(\mathcal{I}/\Delta t + \mathcal{C}_0)y^0, 0, 0, 0}, \underbrace{0, 0, -z_1, 0}, ..., \underbrace{0, 0, -z_{n-1}, 0}, \underbrace{0, 0, -(1+\gamma/\Delta t)z_n, 0} \Big).$$

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

$$G^h(w^h)w^h = f^h \tag{2.10}$$

with the vector $w^h := (w_0^h, w_1^h, ...)$ and $w_i^h := (y_i^h,\ p_i^h,\ \lambda_i^h,\ \xi_i^h)$. Note that the system matrix is obviously a block tridiagonal matrix of the form

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

where $N \in \mathbb{N}$ denotes the number of timesteps, and thus 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 $N+1$ 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

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 $G(w)$. Ignoring for the moment the continuity equations and the boundary conditions, we remember that the coupled KKT system based on the Navier–Stokes equation has the following form:

$$\begin{aligned} y_t - \nu\Delta y + y\nabla y + \nabla p + \tfrac{1}{\alpha}\lambda &= 0 \quad \text{in } \Omega \\ -\lambda_t - \nu\Delta\lambda - y\nabla\lambda + (\nabla y)^t\lambda + \nabla\xi - y &= -z \quad \text{in } \Omega \end{aligned}$$

For two vectors (y, p, λ, ξ) and $(\bar{y}, \bar{p}, \bar{\lambda}, \bar{\xi})$ the Frechét derivative of this system is given by

$$\tilde{\mathcal{F}}(y,p,\lambda,\xi)\begin{pmatrix}\bar{y}\\ \bar{p}\\ \bar{\lambda}\\ \bar{\xi}\end{pmatrix} := \begin{pmatrix} \bar{y}_t - \nu\Delta\bar{y} + (\bar{y}\nabla y + y\nabla\bar{y}) + \nabla\bar{p} + \frac{1}{\alpha}\bar{\lambda} \\ -\bar{\lambda}_t - \nu\Delta\bar{\lambda} - y\nabla\bar{\lambda} + (\nabla y)^t\bar{\lambda} + \nabla\bar{\xi} - \bar{y} - \bar{y}\nabla\lambda + (\nabla\bar{y})^t\lambda \end{pmatrix}.$$

Correspondingly, the Fréchet derivative of the operator $G(w)$ is given by the Newton matrix

$F(w) =$

$$\left(\begin{array}{cccc|cccc|cccc|cccc}
\frac{\mathcal{I}}{\Delta t}+\mathcal{N}_0 & \mathcal{G} & 0 & 0 & 0 & & & & & & & & & & & \\
\mathcal{D} & 0 & 0 & 0 & & 0 & & & & & & & & & & \\
-\mathcal{I}+\mathcal{R}_0 & 0 & \frac{\mathcal{I}}{\Delta t}+\mathcal{N}_0^* & \mathcal{G} & & & -\frac{\mathcal{I}}{\Delta t} & & & & & & & & & \\
0 & 0 & \mathcal{D} & 0 & & & & 0 & & & & & & & & \\
\hline
-\frac{\mathcal{I}}{\Delta t} & & & & \frac{\mathcal{I}}{\Delta t}+\mathcal{N}_1 & \mathcal{G} & \frac{1}{\alpha}\mathcal{I} & 0 & 0 & & & & & & & \\
 & 0 & & & \mathcal{D} & 0 & 0 & 0 & & 0 & & & & & & \\
 & & 0 & & -\mathcal{I}+\mathcal{R}_1 & 0 & \frac{\mathcal{I}}{\Delta t}+\mathcal{N}_1^* & \mathcal{G} & & & -\frac{\mathcal{I}}{\Delta t} & & & & & \\
 & & & 0 & 0 & 0 & \mathcal{D} & 0 & & & & 0 & & & & \\
\hline
 & & & & \ddots & & & & \ddots & & & & \ddots & & & \\
\hline
 & & & & & & & & -\frac{\mathcal{I}}{\Delta t} & & & & \frac{\mathcal{I}}{\Delta t}+\mathcal{N}_n & \mathcal{G} & \frac{1}{\alpha}\mathcal{I} & 0 \\
 & & & & & & & & & 0 & & & \mathcal{D} & 0 & 0 & 0 \\
 & & & & & & & & & & 0 & & -(1+\frac{\gamma}{\Delta t})\mathcal{I}+R_n & 0 & \frac{\mathcal{I}}{\Delta t}+\mathcal{N}_n^* & \mathcal{G} \\
 & & & & & & & & & & & 0 & 0 & 0 & \mathcal{D} & 0
\end{array}\right)$$

with the additional operator $\mathcal{R}_i v := \mathcal{R}(\lambda_i)v := -(v\nabla)\lambda_i + (\nabla v)^t\lambda_i$ for all velocity vectors v.

3 The multigrid and the Newton solver

The KKT-system represents a boundary value problem in the space-time cylinder. It is shown e.g. in [5] 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 cylinder for either the state or the adjoint state. This indicates that multigrid could be used to solve 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 loop

To treat the nonlinearity in the underlying Navier–Stokes equation, 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 defect correction loop,

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

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. $C(w^h)^{-1}$ is applied by the following space-time multigrid method.

3.2 The multigrid solver

To formulate the multigrid solver, let Ω_1, ..., Ω_k for $k \in \mathbb{N}$ be a conformal hierarchy of triangulations of the domain Ω, with Ω_{i+1} stemming from a regular refinement of Ω_i (i.e. new vertices, cells and edges are generated by connecting opposite midpoints of edges). We use V_1, ..., V_k to refer to the different Finite Element spaces in space built upon these meshes. Furthermore, let T_1, ..., T_k be a hierarchy of decompositions of the time interval $[0, T]$, where each T_{i+1} stems from T_i by bisecting each time interval. For each i, the above discretisation in space and time yields a solution space $W_i = V_i \times T_i$ and a space-time system

$$G^i w^i = f^i, \qquad i = 1, ..., k$$

of the form (2.10) with $f^k = f^h$, $w^k = w^h$ and $G^k = G^h$ identifying the discrete right hand side, the solution and system operator on the finest level, respectively.

To describe the multigrid solver, we need prolongation and restriction operators. Let us denote by $I : W_i \to W_{i+1}$ the prolongation and by $R : W_i \to W_{i-1}$ the corresponding restriction. Furthermore, let $S : W_i \to W_i$ define a *smoothing* operator (see the following sections for a definiton of these operators) and let us denote with $NSMpre$, $NSMpost \in \mathbb{N}$ 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 [1, 9, 22].

Algorithm 1 Space-time multigrid

function SPACETIMEMULTIGRID(w;f;k)
  if ($k = 1$) **then**
    return $(G^k)^{-1} f$ ▷ coarse grid solver
  end if
  while (not converged) **do**
    $w \leftarrow S(G^k, w, f, NSMpre)$ ▷ presmoothing
    $d \leftarrow R(f - G^k w)$ ▷ restricion of the defect
    $w \leftarrow w + I(\text{SPACETIMEMULTIGRID}(0; d; k-1))$ ▷ coarse grid correction
    $w \leftarrow S(G^k, w, f, NSMpost)$ ▷ postsmoothing
  end while
  return w ▷ solution
end function

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 [10] and space prolongation/restriction. Let $k \in \mathbb{N}$ be the space level. Then, we denote by $I_S : V_k \to V_{k+1}$ the prolongation in space and $R_S : V_{k+1} \to V_k$ the corresponding restriction. The prolongation for a space-time vector $w^k = (w_0^k, ..., w_N^k)$ can be written as:

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

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

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

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 [14, 18].

3.4 Smoothing operators

The special matrix structure of the global space-time matrix (2.10) allows to define iterative smoothing operators for this algorithm based on defect correction. Note that every smoother usually can also be used as coarse grid solver to solve the equation $(G^k)^{-1} f$ in the first step of the algorithm by replacing the fixed number of iterations by a terminating condition depending on the residuum.

We introduce two basic iterative block smoothing algorithms. Let $\omega \in \mathbb{R}$ be a damping parameter. Then, the special matrix structure suggests the use of a Block-Jacobi method of the following form, see Algorithm 2.

Algorithm 2 Space-time Block-Jacobi smoother

function JACSMOOTHER(G^k,w,f,NSM)
  for $j = 0$ to NSM **do**
    $d \leftarrow f - G^k w$ ▷ Defect
    for $i = 0$ to N **do**
      $d_i \leftarrow (G_i^k)^{-1} d_i$ ▷ Block-Jacobi preconditioning
    end for
    $w \leftarrow w + \omega d$
  end for
  return w ▷ Solution
end function

Algorithm 3 Forward-Backward Block-SOR smoother

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

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 $x_{-1} := x_{N+1} := 0$, $\tilde{M}_0 := \hat{M}_N := 0$.) Again, let $\omega \in \mathbb{R}$ be a damping parameter; for $\omega = 1$ the algorithm reduces to a standard block Gauß-Seidel algorithm. In contrast to Block-Jacobi, this algorithm respects the solutions backward and forward in time, so a stronger time coupling is reached without more additional costs.

Note that the key feature of both algorithms is the solution of the system $G_i^k c_i = d_i$ which means to solve fully coupled KKT system in one time step. The full space-time algorithm therefore reduces to an algorithm in space. It can be carried out without the necessity of setting up and saving the whole space time matrix in memory. The system $G_i^k c_i = d_i$ is a coupled saddle point problem for primal and dual velocity and pressure. For solving it, one can use e.g. direct solvers (as long as the number of unknowns in space is not too large) or sophisticated techniques from computational fluid dynamics, namely a space multigrid with Pressure-Schur-Complement based smoothers. A typical approach can be seen in the next section.

3.5 Coupled multigrid solvers in space

As mentioned above, in each time step a system of the form $G_i^k c_i = d_i$ must be solved, e.g. with a multigrid solver in space. Since the used prolongation and restriction operators based on the applied Finite Element spaces are standard and well known (see e.g. [1, 4, 9, 18, 22]), we restrict here to a short introduction into the pressure Schur complement ('PSC') approach for CFD problems (see also [17, 20, 21]) which is used as a smoother, acting simultaneously on the primal and dual variables. A complete overview and in-depth description of the operators and smoothers will be given in [15].

To formulate the corresponding algorithm, we first introduce some notations. Let $iel \in \mathbb{N}$ denote the number of an arbitrary element in the mesh. On this mesh, a linear system $Ax = b$ is to be solved; in our case, this system can be written in the form

$$\begin{pmatrix} A^{\text{primal}} & M^{\text{dual}} & B & 0 \\ M^{\text{primal}} & A^{\text{dual}} & 0 & B \\ B^T & 0 & 0 & 0 \\ 0 & B^T & 0 & 0 \end{pmatrix} \begin{pmatrix} y \\ \lambda \\ p \\ \xi \end{pmatrix} = \begin{pmatrix} b_y \\ b_\lambda \\ b_p \\ b_\xi \end{pmatrix}$$

which is a typical saddle point problem for primal and dual variables, with A^{primal}, A^{dual} velocity submatrices, M^{primal} and M^{dual} coupling matrices between the primal and dual velocity and B and B^T clustering the gradient/divergence matrices.

Now let $I(iel)$ identify a list of all degrees of freedom that can be found on element iel, containing numbers for the primal and dual velocity vectors in all spatial dimensions and the primal and 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 $I(iel)$.

This notation allows to formulate the basic PSC smoother in space, providing $\omega \in \mathbb{R}$ to be a damping parameter, see Algorithm 4. 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

Algorithm 4 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

C_{iel}^{-1} by invoking a LU decomposition, e.g. with the LAPACK package [16]. 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 [18]), 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

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

where *diag(·)* refers to the operator taking only the diagonal of a given matrix. The local preconditioner can then be formulated as $C_{iel} := \hat{A}_{I(iel),I(iel)}$. Note that this matrix decouples the primal variables from the dual variables, so that applying $\hat{A}^{-1}_{I(iel),I(iel)}$ decomposes into two independent subproblems for the primal variables (y, p) and the dual variables (λ, ξ). The special feature that the velocity blocks are diagonal allows to efficiently use Schur complement techniques by what this smoother is numerically cheaper than PSCSmootherFull. Nevertheless, the disadvantage of this smoother is the reduced stability. We note that the PSC approach in general also allows to increase the stability by taking larger local systems. Such an approach was carried out e.g. in [17] where the degrees of freedom of multiple adjacent elements were clustered together to form a linear subsystem. For our numerical tests however, such an approach was not necessary. Most of the numerical tests in the later sections were carried out using PSCSmootherDiag except where noted.

4 Numerical examples

In this section we numerically analyse the proposed solver strategy. The nonlinearity is captured by a space-time fixed point and Newton iteration, both preconditioned by the proposed space-time multigrid. We define the following test cases:

4.1 Example (*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 $u(x) = (0, 0)$ for $x \in \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ and $u(x) = (1, 0)$

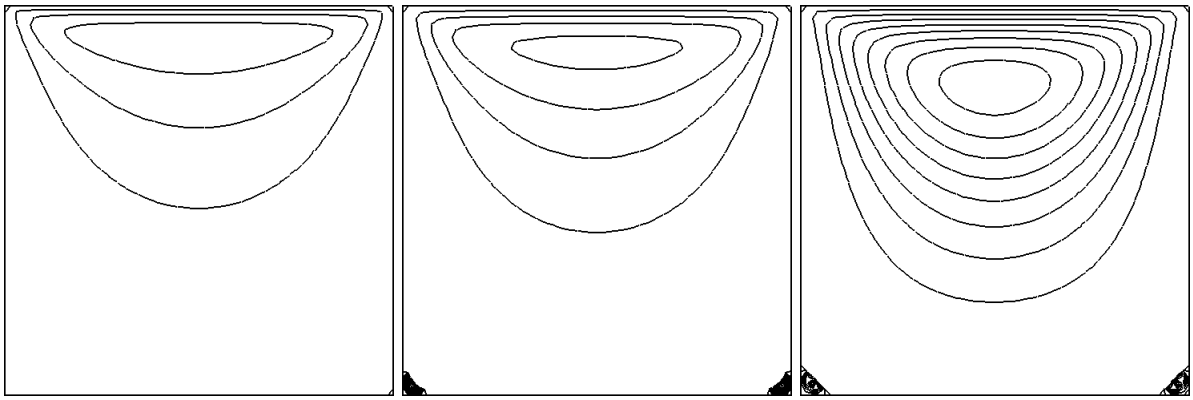

Figure 4.1: *Driven-cavity* example, Stokes (target-) flow. Streamlines of the flow at time $t = 0.5$ (left), $t = 1$ (center) and $t = 8$ (right)

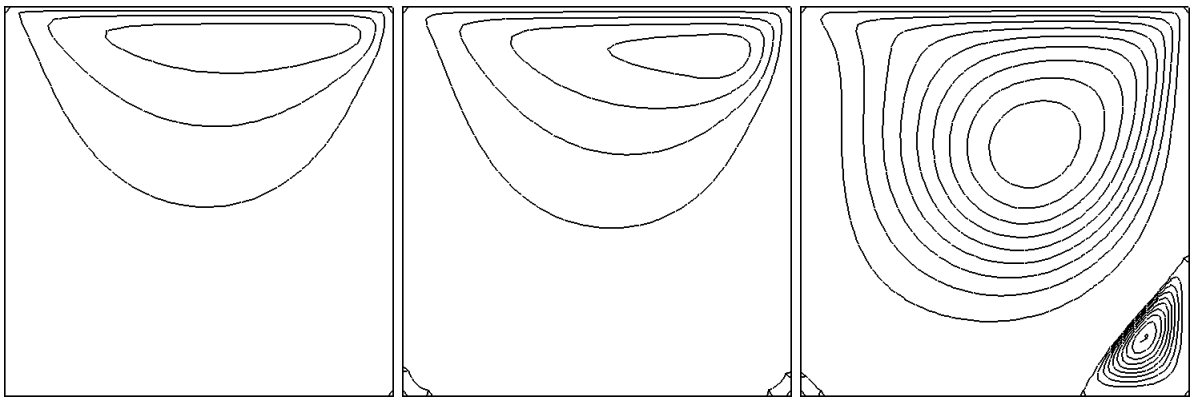

Figure 4.2: *Driven-cavity* example, uncontrolled Navier–Stokes flow. Streamlines of the flow at time $t = 0.5$ (left), $t = 1$ (center) and $t = 8$ (right).

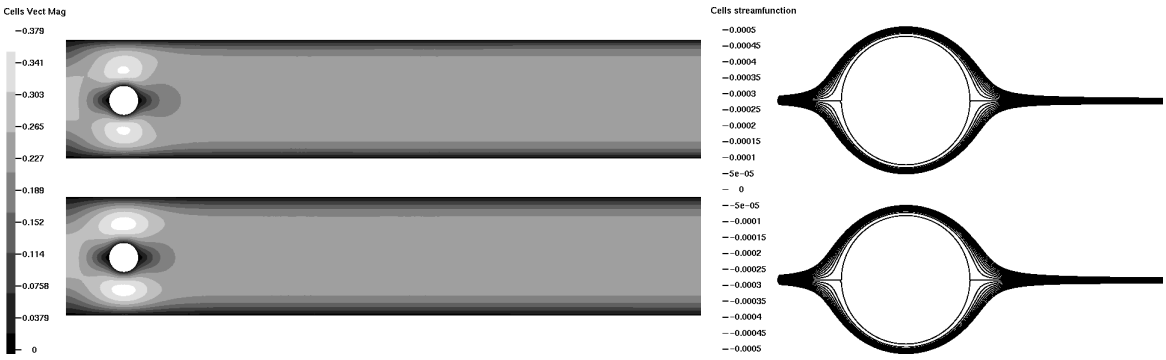

Figure 4.3: *Flow-around-cylinder* example. Stokes (target-) flow at time $t = 0.5$ (top) and $t = 1$ (bottom). Left: Velocity magnitude field. Right: Streamlines around the cylinder.

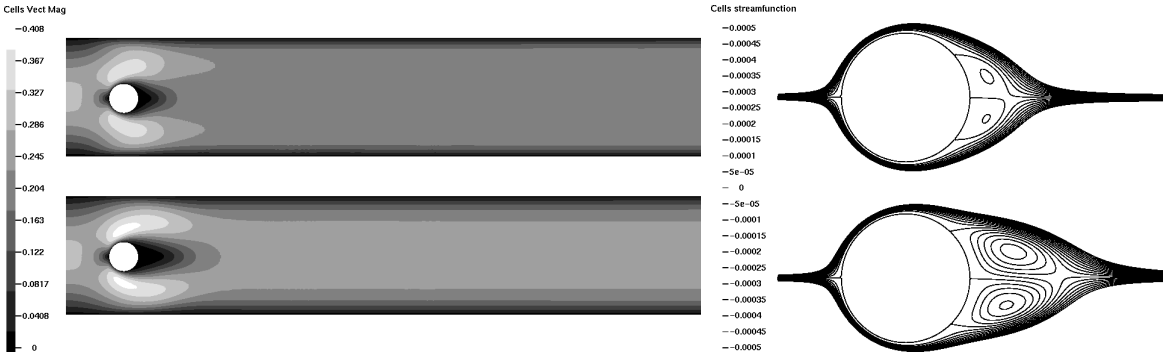

Figure 4.4: *Flow-around-cylinder* example. Uncontrolled Navier–Stokes flow at time $t = 0.5$ (top) and $t = 1$ (bottom). Left: Velocity magnitude field. Right: Streamlines around the cylinder. Note: The nonsymmetry stems from the nonsymmetric position of the cylinder.

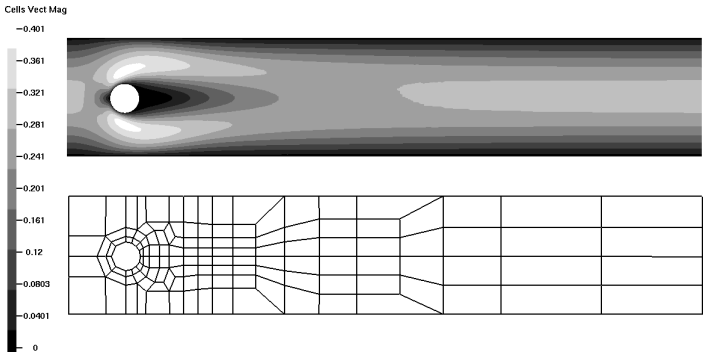

Figure 4.5: Top: Uncontrolled Navier–Stokes flow in the *flow-around-cylinder* example. Velocity magnitude, fully developed stationary flow. Bottom: Underlying mesh, coarse grid.

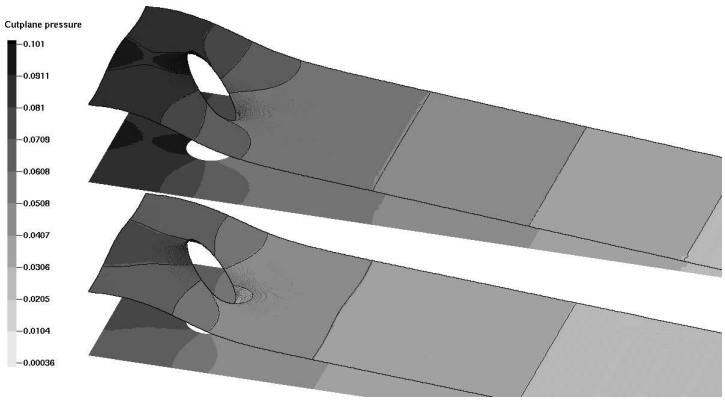

Figure 4.6: *Flow-around-cylinder* example, 3D topography map of the pressure. Stokes (target-) flow at time $t = 0.5$ (top) and $t = 1$ (bottom).

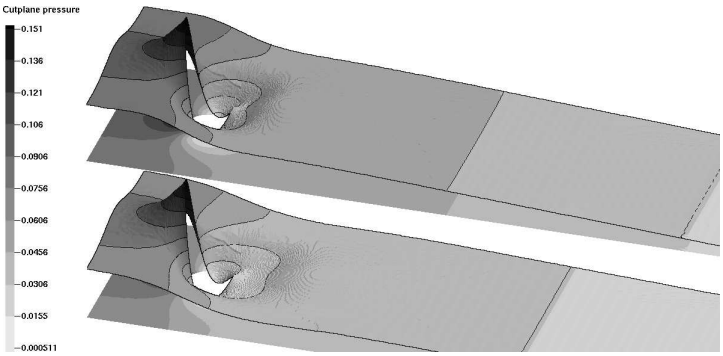

Figure 4.7: *Flow-around-cylinder* example, 3D topography map of the pressure. Uncontrolled Navier–Stokes flow at time $t = 0.5$ (top) and $t = 1$ (bottom).

Driven-cavity:		simulation		optimisation	
Δt	h	#DOF space	#DOF total	#DOF space	#DOF total
1/8	1/8	352	3 168	704	6 336
1/16	1/16	1344	22 848	2 688	45 696
1/32	1/32	5248	173 184	10 496	346 368
1/64	1/64	20736	1 347 840	41 472	2 695 680

Flow-around-cylinder:		simulation		optimisation	
Δt	Space-Lv.	#DOF space	#DOF total	#DOF space	#DOF total
1/10	2	2704	29 744	5 408	59 488
1/20	3	10608	222 768	21 216	445 536
1/40	4	42016	1 722 656	84 032	3 445 312
1/80	5	167232	13 545 792	334 464	27 091 584

Table 4.1: Problem size we use in our numerical tests in the simulation and the optimisation on different refinement levels. '#DOF space' describes the number of degrees of freedom in space, i.e. in every timestep. '#DOF total' refers to the total number of degrees of freedom on the whole space-time cylinder including the initial condition.

for $x \in \Gamma_4$. The coarse grid consists of only one quadratic element. As time domain we define the interval $[0,T]$ with $T = 1$. The viscosity parameter is set to $\nu = 1/400$. On the whole space-time cylinder, we generate a target flow z by computing a simulation of the nonstationary *Stokes* equation with right hand side $f := 0$, viscosity parameter $\nu = 1/400$ as well and the initial flow at $t = 0$ at rest. A picture of the streamlines of the target flow can be seen in Figure 4.1 while Figure 4.2 shows the uncontrolled Navier–Stokes flow. For better visualisation, the streamlines of the big vortex correspond to the value interval $[-0.1, 0]$ while the streamlines of the small vortices in the bottom corners highlight the value interval [0, 1e-6] in figure 4.1 and [0, 4e-4] in Figure 4.2.

4.2 Example (Flow-around-cylinder configuration)**.** As spatial domain, we prescribe a rectangle without an inner cylinder $\Omega := [0, 2.2] \times [0, 0.41] \setminus B_r(0.2, 0.2)$, $r = 0.05$, cf. [18]. 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 $\Gamma_5 := \partial B_r(0.2, 0.2)$. As boundary conditions, we define $u(x) := (0, 0)$ for $x \in \Gamma_2 \cup \Gamma_4 \cup \Gamma_5$. On Γ_3 we prescribe do-nothing boundary conditions while on Γ_1 a parabolic inflow profile with maximum velocity $U_{\max} := 0.3$ is used. The time interval for this test case we define as $[0,T]$ with $T = 1$. To generate a target flow z, we compute a nonstationary simulation using the *Stokes equation*, the initial flow at $t = 0$ at rest. The right hand side is set to $f := 0$ and viscosity parameter to $\nu = 1/1000$ (resulting in Re=20). Figure 4.3 shows the velocity field of the reference Stokes flow as well as the streamlines around the cylinder at time $t = 0.5$ and $t = 1$, while in Figure 4.4 visualises those of a corresponding simulation with the Navier–Stokes equation. The fully developed stationary Navier–Stokes flow as well as the underlying mesh can be seen in Figure 4.5.

4.3 Example (Flow-around-cylinder with pulsating inflow)**.** The spatial domain is the same as in Example 4.2. We use the time cylinder $[0,T]$ with $T = 8$ and a viscosity parameter $\nu = 1/1000$. As boundary conditions, we again define $u(x) := (0, 0)$ for $x \in \Gamma_2 \cup \Gamma_4 \cup \Gamma_5$ and do-nothing boundary conditions on $\Gamma_1 \cup \Gamma_3$. To generate a target flow z, we define on Γ_1 an oscillating Dirichlet inflow boundary condition as parabolic profile with maximum inflow velocity $U_{\max}(t) := 0.15(1 + \sin(\frac{(t+3)}{2}\pi))/2$ and compute a full Navier–Stokes simulation with $f := 0$ as right hand side, the initial flow at $t = 0$ at rest.

All problems are tested for a various refinements in space and time. Table 4.1 lists the number of degrees of freedom in space and time for the different refinement levels we analyse based on Example 4.1 and 4.2, for a pure forward simulation as well as for optimal control. Discretisation in space is carried out with the $\tilde{Q}_1/Q_0$ finite element pair [18] while implicit backward Euler is used for the time discretisation. All computations are carried out on an AMD Opteron 64 dual core machine with 2.8 GHz and 8 GB RAM running on SUSE Linux.

4.4 Tests with the *Driven-cavity* example

In the first couple of tests we analyse the behaviour of our solver when being applied to the *Driven-cavity* example. For our tests, we 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. For simplicity, we choose $\Delta t = h = 1.0$, although any other relation between Δt and h

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: 1.) We calculate a pure simulation with a fully implicit Navier–Stokes solver in time. In each timestep the norm of the residual was reduced by $\varepsilon_{\mathrm{SimNL}} = 10^{-5}$. The linear multigrid subsolver in each nonlinear iteration reduces the norm of the residual by $\varepsilon_{\mathrm{SimMG}}$. 2.) We calculate an optimal control problem with the target flow as specified above. The nonlinear space-time solver reduces 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 reduce the norm of the residual by $\varepsilon_{\mathrm{SpaceMG}}$.

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 $\Delta t = h = 1/16$, the convergence criterion of the innermost solver to $\varepsilon_{\mathrm{SpaceMG}} = 10^{-2}$; this criterion was already proven to be useful in [11]. The smoother in space is BiCGStab with PSCSmootherDiag preconditioner, the space-time smoother FBSORSmoother($\omega = 0.9$). Table 4.2 shows the norms of the residuals in the nonlinear steps, for the standard fixed point solver as well as for the Newton solver and for different settings of the convergence criterion of our space-time multigrid. The Newton-solver shows the typical quadratic convergence behaviour in the last iterations when being used with $\varepsilon_{\mathrm{OptMG}} = 10^{-6}$, although it does hardly need less nonlinear steps, as one would expect, due to the global convergence criterion. The global fixed point shows no strong dependence on $\varepsilon_{\mathrm{OptMG}}$ but needs of course more nonlinear steps than the Newton solver.

The next test analyses the influence of the innermost stopping criterion $\varepsilon_{\mathrm{SpaceMG}}$. For the two space-time smoothers JACSmoother($\omega = 0.7$, NSM $= 4$) and FBSORSmoother($\omega = 0.9$, NSM $= 1$) we fix the convergence criterion of the space-time multigrid to $\varepsilon_{\mathrm{OptMG}} = 10^{-6}$

	fixed point		Newton	
Step \ $\varepsilon_{\mathrm{OptMG}}$	10^{-2}	10^{-6}	10^{-2}	10^{-6}
0	5.59E-04	5.59E-04	5.59E-04	5.59E-04
1	1.35E-04	1.35E-04	9.90E-05	9.81E-05
2	3.43E-05	3.34E-05	3.18E-06	2.99E-06
3	9.80E-06	9.10E-06	7.95E-09	2.70E-09
4	3.01E-06	2.69E-06	3.91E-11	
5	9.91E-07	8.51E-07		
6	3.33E-07	2.73E-07		
7	1.09E-07	8.52E-08		
8	3.43E-08	2.54E-08		
9	1.03E-08	7.25E-09		
10	2.98E-09	1.98E-09		

Table 4.2: *Driven-cavity* example. Behaviour of the space-time fixed point and Newton solver for different settings of $\varepsilon_{\mathrm{OptMG}}$ for the linear space-time subproblems.

		JACSmoother(NSM=4)			FBSORSmoother(NSM=1)		
Nonl. Solv.	$\varepsilon_{\text{SpaceMG}}$	#NL	#MG	Time	#NL	#MG	Time
fixed point	10^{-1}	10	110	590.0	10	69	285.9
	10^{-2}	10	110	617.0	10	68	296.3
	10^{-3}	10	110	735.8	10	68	349.5
	10^{-6}	10	110	1207.3	10	68	528.3
Newton	10^{-1}	3	34	219.8	3	23	99.8
	10^{-2}	3	34	228.1	3	23	102.7
	10^{-3}	3	34	322.7	3	23	123.6
	10^{-6}	3	34	541.7	3	23	179.0

Table 4.3: The global nonlinear solver is not influenced by the convergence criterion of the multigrid solver in space. '#NL' describes the number of iterations in the nonlinear solver, '#MG' the number of iterations in the space-time multigrid solver, 'Time' the computational time in seconds; *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
1/8	1/8	6	13	4	12	6	46	3	27
1/16	1/16	10	20	4	11	10	68	3	23
1/32	1/32	8	16	4	10	8	53	3	21
1/64	1/64	6	12	4	8	6	40	3	18

Table 4.4: Number of iterations in the nonlinear (#NL) and linear (#MG) space-time solver, for various levels of refinement and different settings for $\varepsilon_{\text{OptMG}}$. The smoother is FBSORSmoother($\omega = 0.9$); *Driven-cavity* example.

Nonl. Solv.	Δt	h	#NL	#MG
Newton	1/16	1/64	3	6
	1/32	1/64	3	6
	1/64	1/64	4	8

Table 4.5: Number of iterations in the nonlinear (#NL) and linear (#MG) space-time solver, for various levels of refinement in time. $\varepsilon_{\text{OptMG}}$ was fixed to 10^{-2}. The smoother is FBSORSmoother($\omega = 0.9$); *Driven-cavity* example.

and calculate the fixed point and Newton iteration for different settings of $\varepsilon_{\text{SpaceMG}}$. As one can see from Table 4.3, the solver behaves very robust against this parameter. We choose $\varepsilon_{\text{SpaceMG}} = 10^{-2}$ for all later tests. Furthermore one can see, that JACSmoother is by far less efficient than FBSORSmoother: Although there are 4 times more smoothing steps used, the iteration needs more iterations and more computational time. Therefore, we drop JACSmoother in our future tests.

Table 4.4 reveals the dependence of the nonlinear space-time solver on the convergence criterion of the space-time multigrid solver. The number of linear and nonlinear iterations stays constant[1] upon increasing the resolution of the space-time mesh which shows 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 [5].

Furthermore, for a convergence criterion of $\varepsilon_{\text{OptNL}} = 10^{-5}$, a reduction of $\varepsilon_{\text{OptMG}}$ from 10^{-2} to 10^{-6} has no visible influence to the nonlinear solver but increases the number of linear iterations by a factor of approx. 3. Thus we take $\varepsilon_{\text{OptMG}} = 10^{-2}$ for all further numerical tests.

Table 4.5 finally shows what happens when Δt is changed against h. We fixed $h = 1/64$ and reduced Δt. We cannot expect the number of iterations of the space-time multigrid to stay constant in this case as this is only guaranteed if Δt and h changes simultaneously upon refinement, but as can be seen in the table, increasing Δt for a fixed h does not have a great influence to the solver. This effect suggests that it is feasible to use higher order timestepping techniques that allow to use large timesteps, but as higher order timestepping techniques is a topic of its own, we will not discuss this topic in the scope of this paper.

Optimisation vs. Simulation

In the following tests we want to 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 4.6 depicts the result of a set of forward simulations for various settings of Δh and t. As nonlinear solver in each time step of the simulation, we use on the one hand a simple nonlinear fixed point iteration, on the other hand a Newton iteration. The linear solver used here was multigrid in space with a PSCSmootherDiag-type smoother.

The columns ⊘NL and ⊘MG in this table describe the average number of linear/nonlinear iterations per time step. When comparing these numbers with the number of iterations of the nonlinear/linear space-time solver in Table 4.4 one can see that they are rather similar. Focusing on the Newton iteration, the space-time Newton solver needs about the same number of global nonlinear iterations like the nonlinear solver in each timestep of the simulation. The total number iterations of the space-time multigrid solver in the optimisation solver differs to the total number of iterations of the space multigrid solver by a factor of approx. 2-3, which means that the effort for both, the simulation and the optimisation, grows with same complexity when increasing the problem size. This fact encourages us to directly compare the

[1] We note that there would of course be a difference in #NL and #MG if we choose a stronger convergence criterion than $\varepsilon_{\text{OptNL}} = 10^{-5}$ in combination with the Newton solver; this is already indicated by the size of the last nonlinear residuum in Table 4.2.

Nonl. Solv.	Δt	h	#NL	#MG	⌀NL	⌀MG
fixed point	1/8	1/8	51	165	6	21
	1/16	1/16	94	525	6	33
	1/32	1/32	159	1009	5	32
	1/64	1/64	265	1925	4	30
Newton	1/8	1/8	25	79	3	10
	1/16	1/16	50	302	3	19
	1/32	1/32	99	762	3	24
	1/64	1/64	196	1669	3	26

Table 4.6: Total and mean number of nonlinear and linear iterations per timestep for a forward simulation on different levels of refinement; *Driven-cavity*-example.

Nonl. Solv.	Δt	h	T_{sim}	T_{opt}	$\frac{T_{\text{opt}}}{T_{\text{sim}}}$
fixed point	1/8	1/8	0.46	14.21	31.0
	1/16	1/16	2.20	74.83	34.0
	1/32	1/32	15.76	351.21	22.3
	1/64	1/64	132.43	1568.97	11.8
Newton	1/8	1/8	0.36	14.25	39.2
	1/16	1/16	2.27	48.14	21.2
	1/32	1/32	18.73	301.99	16.1
	1/64	1/64	233.34	1348.12	5.8

Table 4.7: Comparison of the execution time for a simulation and a corresponding optimisation; *Driven-cavity*-example.

Nonl. Solv.	Δt	h	#NL	#MG	T_{opt}	$\frac{T_{\text{opt}} \text{ with } T=8}{T_{\text{opt}} \text{ with } T=1}$
fixed point	1/8	1/8	6	14	125	5.41
	1/16	1/16	11	23	878	11.37
	1/32	1/32	8	16	2946	7.78
	1/64	1/64	6	12	15127	9.06
Newton	1/8	1/8	4	14	244	15.04
	1/16	1/16	4	12	473	8.33
	1/32	1/32	4	11	2600	8.83
	1/64	1/64	4	8	13879	8.90

Table 4.8: *Driven-cavity*-example on the longer time interval $[0, T]$ with $T = 8$. The optimisation takes approx. $8\times$ longer than the optimisation on the time interval $[0, 1]$.

time that was necessary for the simulation against the time necessary for the corresponding optimisation.

Table 4.7 compares the different execution times of the simulation and optimisation solver. While the fraction of the execution times is still inaccurate for 'small' mesh resolutions (Δt, $h \leq 1/16$) in conjunction with the fixed point method, 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 $C \approx 10-30$, even being better for higher mesh resolutions. We note here that the high factors for small meshes are merely an effect of acceleration due to cache effects and can be expected: A simulation with only some hundreds of unknowns is able to rather completely run in the cache of a modern PC, while a higher mesh resolution leads to cache misses and therefore to a slowdown in the computational speed. The optimisation of a nonstationary PDE furthermore has not only twice as many unknowns in space but also to simultaneously work on all unknowns in space and time. Therefore, such an optimisation problem can hardly exploit any cache effects, even for problem sizes where the corresponding simulation works still in-cache. A more detailed analysis and exploitation of the computer cache and its effects can be found in [19].

Optimisation on a longer time interval

The above test case was prototypically carried out on a time interval $[0, T]$ with $T = 1$. To test whether there is any impact onto the solver for longer time intervals, we modified test problem 4.1 to calculate nonstationary target Stokes flow on the time interval $[0, T]$ with $T = 8$, the initial flow at rest, and repeat the optimisation for different refinement levels in space and time. From Table 4.8 one can see that a longer time interval has obviously no impact to the global solver. The number of linear and nonlinear iterations are similar. The time that was necessary for the optimisation is now roughly $8 - 9\times$ longer in comparison to the optimisation carried out on the time interval $[0, 1]$. This factor represents exactly the expected behaviour as the problem has now $8\times$ more unknowns.

4.5 Tests with the *flow-around-cylinder* example

***Flow-around-cylinder* with stationary inflow**

In a similar way as above, we now carry out a set of tests for the more complicated *flow-around-cylinder* problem. Our basic mesh (marked as 'level 1') is the one shown in Figure 4.5, regular refinement in space leads to higher mesh levels 2..5. For the time discretisation, we choose $N = 5$ timesteps on time level 1 which leads to 10, 20, 40 and 80 timesteps for all higher time levels. On these space-time-levels, we calculate a simulation and a corresponding optimisation, using the nonstationary Stokes flow from above as target flow. The convergence criteria for the solvers are defined as $\varepsilon_{\text{SimNL}} = \varepsilon_{\text{OptNL}} = 10^{-5}$, $\varepsilon_{\text{SimMG}} = \varepsilon_{\text{OptMG}} = \varepsilon_{\text{SpaceMG}} = 10^{-2}$ and we again focus on the difference in the execution time between the simulation and a corresponding optimisation. Figure 4.8 shows a picture of the velocityand streamline field, Figure 4.9 the pressure of the controlled flow at $t = 0.5$ and $t = 1.0$. The velocity field and the streamlines are similar to the target flow, the pressure peaks around the cylinder are reduced

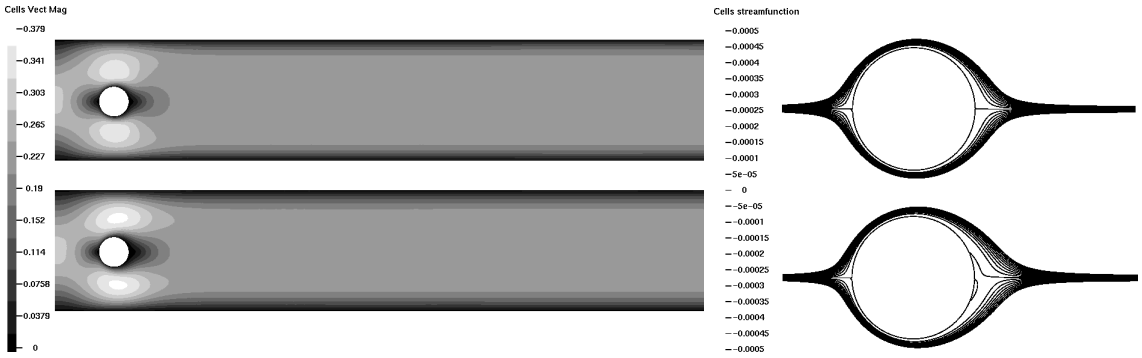

Figure 4.8: *Flow-around-cylinder* example. Velocity magnitude (left) and streamlines around the cylinder (right) controlled solution at $t = 0.5$ (top) and $t = 1$ (bottom).

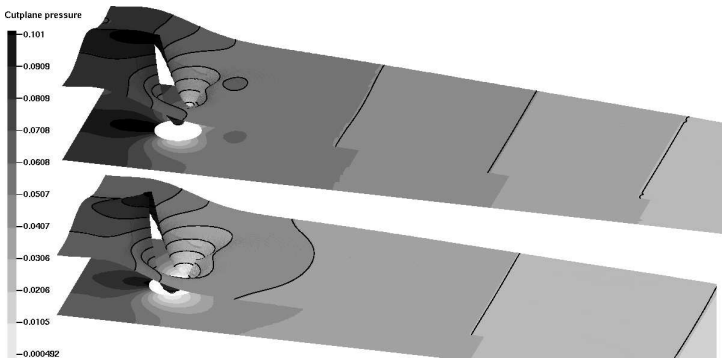

Figure 4.9: *Flow-around-cylinder* example. 3D topography of the pressure. Controlled solution at $t = 0.5$ (top) and $t = 1$ (bottom).

			Simulation				Optimisation	
Nonl. Solv.	Δt	Space-Lv.	#NL	#MG	⌀NL	⌀MG	#NL	#MG
fixed point	1/20	3	100	486	5	24	5	24
	1/40	4	167	946	4	24	5	15
	1/80	5	309	1900	4	24	4	11
Newton	1/20	3	63	312	3	16	4	24
	1/40	4	123	709	3	18	4	14
	1/80	5	246	1589	3	20	4	13

Table 4.9: Total and mean number of nonlinear and linear iterations per timestep for a forward simulation and a corresponding optimisation on different levels of refinement; *flow-around-cylinder* example.

Nonl. Solv.	Δt	Space-Lv.	T_{sim}	T_{opt}	$\frac{T_{\text{opt}}}{T_{\text{sim}}}$
fixed point	1/20	3	22.1	739.27	33.5
	1/40	4	167.9	3509.51	20.9
	1/80	5	1721.7	26741.82	15.5
Newton	1/20	3	27.0	852.33	31.6
	1/40	4	209.6	4037.21	19.3
	1/80	5	2227.1	37217.81	16.7

Table 4.10: Comparison of the execution time for a simulation and a corresponding optimisation. *Flow-around-cylinder* example.

		fixed point		Newton	
Δt	Space-Lv.	#NL	#MG	#NL	#MG
1/20	4	5	13	3	11
1/40	4	5	21	4	21
1/80	4	5	28	4	27

Table 4.11: *Flow-around-cylinder* example. Convergence behaviour of the solver for fixed space-level and different Δt.

by approx. 30% in comparison to the uncontrolled flow.

Table 4.9 depicts the total number of nonlinear (#NL) and linear (#MG) iterations as well as the mean number of nonlinear and linear iterations per timestep for both, the optimisation and the simulation. 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. Table 4.10 again confirms that the execution time of the simulation and the optimisation[2] differs by only a constant factor, which is better for higher levels of refinement. The table indicates a factor $C \approx 15 - 30$.

Table 4.11 now analyses the effect to the solver when modifying Δt against h. We fixed the space level to 4 and used 20, 40 and 80 timesteps. The nonlinear solver is obviously not influenced by this modification. Nevertheless there is a slight impact to the linear solver, which is even a little bit worse when the timestep size is reduced. The fact that larger timesteps do not influence the solver, as it was the case in the *Driven-cavity* example, is therefore not correct for general grids with a large difference in the size between the smallest and largest cells in the mesh.

We continue with a numerical comparison of the solutions we obtained via simulation and

[2]In the table, the execution time 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 $\varepsilon_{\text{OptNL}} < 10^{-5}$ the Newton and the fixed point method need approximately the same number of linear/nonlinear iterations.

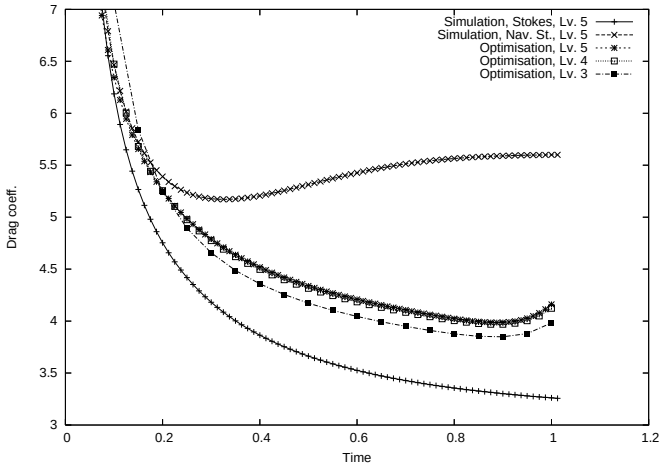

Figure 4.10: Drag coefficients in the *flow-around-cylinder* example; $\alpha = 0.01$, $\gamma = 0$.

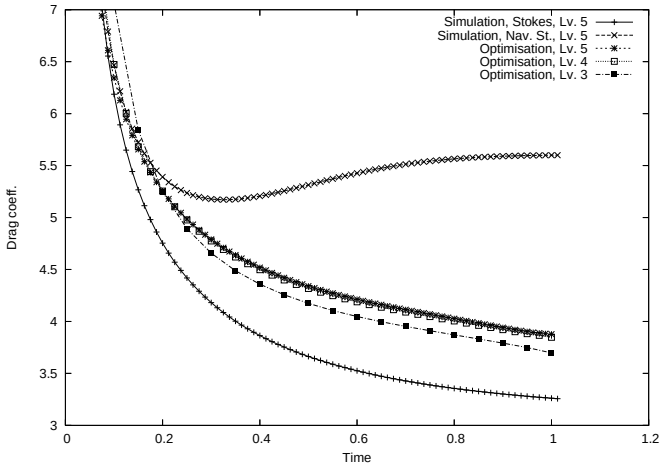

Figure 4.11: Drag coefficients in the *flow-around-cylinder* example, $\alpha = 0.01$, $\gamma = 0.1$.

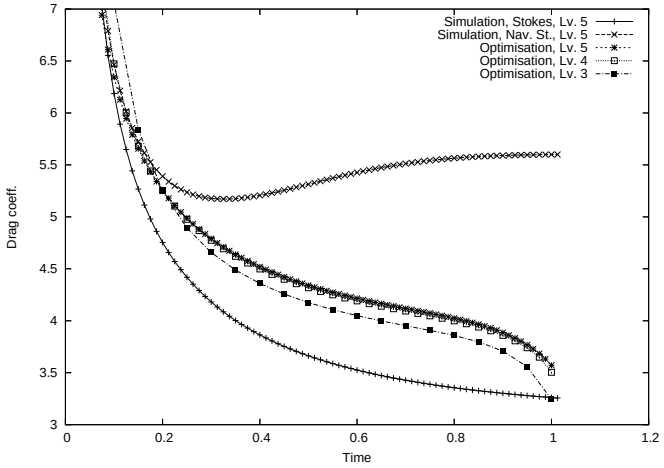

Figure 4.12: Drag coefficients in the *flow-around-cylinder* example, $\alpha = 0.01$, $\gamma = 0.5$.

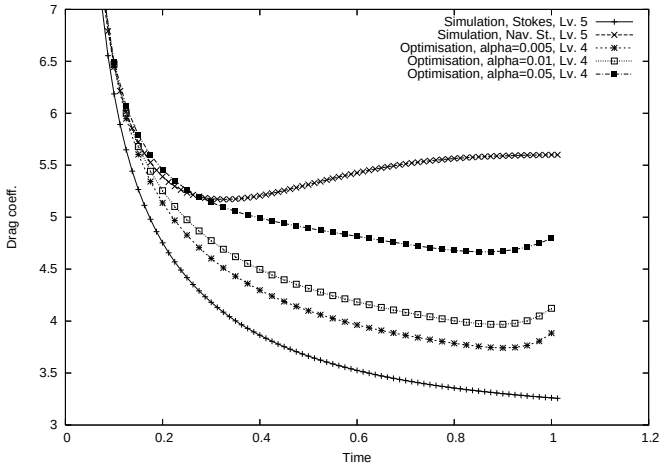

Figure 4.13: Drag coefficients for *flow-around-cylinder*; $\gamma = 0$, $\alpha = 0.005,\ 0.01$ and 0.05.

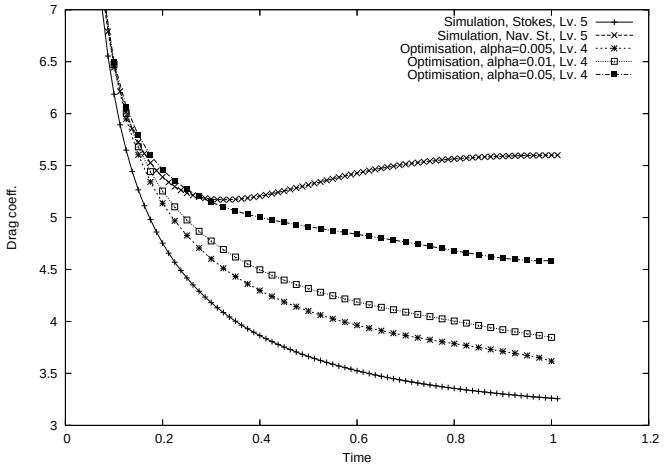

Figure 4.14: Drag coefficients for *flow-around-cylinder*; $\gamma = 0.1$, $\alpha = 0.005,\ 0.01$ and 0.05.

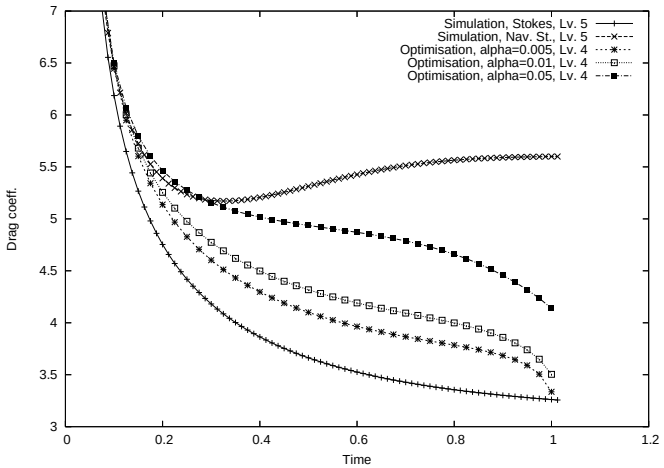

Figure 4.15: Drag coefficients for *flow-around-cylinder*; $\gamma = 0.5$, $\alpha = 0.005,\ 0.01$ and 0.05.

		$\|\|y-z\|\|_{L^2(Q)}$	$\|\|u\|\|_{L^2(Q)}$	$\|\|y(T)-z(T)\|\|_\Omega$	J(y,u)	rel. err. C_D
	Initial	1.39E-01	0.00E+00	1.43E-01	1.06E-02	at $t=1$
$\gamma=0$	$\alpha=0.05$	1.95E-02	7.27E-01	9.28E-03	2.23E-04	47%
	$\alpha=0.01$	1.60E-02	8.43E+00	4.79E-03	1.64E-04	26%
	$\alpha=0.005$	1.48E-02	2.26E+01	3.42E-03	1.41E-04	19%
$\gamma=0.1$	$\alpha=0.05$	1.94E-02	7.54E-01	7.42E-03	2.26E-04	41%
	$\alpha=0.01$	1.60E-02	8.50E+00	2.94E-03	1.65E-04	18%
	$\alpha=0.005$	1.48E-02	2.27E+01	1.85E-03	1.41E-04	11%
$\gamma=0.5$	$\alpha=0.05$	1.93E-02	8.17E-01	4.26E-03	2.32E-04	27%
	$\alpha=0.01$	1.60E-02	8.60E+00	1.30E-03	1.66E-04	7%
	$\alpha=0.005$	1.48E-02	2.28E+01	8.11E-04	1.41E-04	2%

Table 4.12: *Flow-around-cylinder* example. Initial and final values of the functional and its components used for minimisation in the optimisation process. Right: Maximum relative error in C_D at time t=1.

optimisation. For that purpose, we calculate the drag coefficient

$$C_D = \frac{2}{(\frac{2}{3}U_{\max})^2 \cdot 0.1} \int_{\Gamma_5} \nu \frac{\partial y_t}{\partial n} n_y - p n_x \, dS$$

(with $n = (n_x, n_y)$ the normal and $t = (n_y, -n_x)$ the tangential vector of Γ_5) around the circle in every timestep, for the results of the simulation, the reference flow and multiple refinement levels of the optimisation. Figure 4.10 visualises these drag coefficients; the topmost curve shows the simulation with Navier–Stokes and the bottommost the target flow. The curves obtained by optimisation follow the curve of the reference flow and converge with increasing space-time level. The maximum relative error on the time interval $[0.1, 1]$ is approx. 30% and appears at the end of the time interval, where (because of $\gamma = 0$) the terminal condition has no influence to the flow. On the time interval $[0.1, 0.8]$ the relative error is less than 20%.

The influence of the terminal condition introduced by different values for γ can be seen in Figure 4.11 and 4.12. A value of $\gamma = 0.1$ seems to be appropriate for this test example. A value $\gamma = 0.5$ shows an even better aggreement with the reference flow but introduces a strong bending in the curve of the drag coefficients (which we note was leading to worse convergence rates of all linear solvers, see also Table 4.13). This effect is obviously independent of the refinement of the space-time mesh!

In a last set of tests with this configuration we want to clarify whether there is a strong dependence of the solution when both α and γ are modified. For that purpose, we at first calculate the values of the functional $J(\cdot)$ and its components on space-time level 4 for $\alpha = 0.005$, 0.01 and 0.05 and $\gamma = 0$, 0.1 and 0.5. Table 4.12 compares the results. The table shows the initial values for $J(\cdot)$ and its components, the final values computed by the result of the of the nonlinear iteration and the maximum relative error in C_D to the drag of the target flow at the end of the time interval at time t=1. In all cases, the errors are reduced, usually by an order of magnitude. A lower value of α results in a better approximation, indicated by a lower $||y-z||_{L^2(Q)}$ value. The norm $||y-z||_{L^2(Q)}$ is rather insensitive to the choice of γ, but γ

has a major influence to the error at the end of the time interval $||y(T) - z(T)||_\Omega$. $\alpha = 0.005$ and $\gamma = 0.5$ result in the best overall values.

To show the effect of modifying α and γ on the solution more clearly, we visualise the calculated drag coefficients calculated in this test. For a fixed γ, Figure 4.13, 4.14, and 4.15, depict the behaviour of the drag coefficients if α is changed. One can see a slight dependence on α: For $\gamma = 0$, all curves bend the same way up to the reference solution calculated with the Navier–Stokes equation while for $\gamma = 0.5$ all curves bend down. For smaller α, the bending is slightly stronger and begins later than for larger values of α. Nevertheless the dependence is weak: For $\gamma = 0.1$ there is nearly no bending, independent of the value of α. We can therefore state that α and γ can be chosen rather independent without noticable influence to each other with respect to the solution.

The impact on the solver when modifying α and γ can be seen in Tablle 4.13. Although the nonlinear solver is not influenced, larger values for α and γ have obviously an impact onto the space-time solver which needs more iterations to solve the linear subproblems. Furthermore, the difficulty for solving the linear subproblems in space increases: For $\alpha = 0.005$ and $\gamma = 0.5$ we had to use the stronger PSCSmootherFull smoother in space to calculate the corresponding optimisation problem. For $\gamma = 0.1$ the convergence was the best, the number of linear iterations being independent of the value of α. A deeper analysis of the impact of α and γ to the solver s a subject of future research and will help to improve the efficiency and robustness of the solver components.

α	0.05		0.01		0.005	
γ	#NL	#MG	#NL	#MG	#NL	#MG
0	5	15	5	15	5	15
0.1	5	15	5	15	5	15
0.5	5	15	5	40	5	79

Table 4.13: *Flow-around-cylinder* example. Number of nonlinear and linear steps of the solver when modifying α and γ.

Flow-around-cylinder with pulsating inflow

The last set of tests focuses on Example 4.3 to analyse the behaviour of the solver as well as the accuracy of the solution in a fully nonstationary case with oscillating boundary conditions. The target flow z was generated by a simulation of the Navier–Stokes equation with pulsating Dirichlet inflow boundary conditions. For the optimisation, we now prescribe Neumann boundary conditions on both ends of the domain, i.e. Γ_1 and Γ_3, so the uncontrolled flow would be at rest. Introducing the control therefore aims to imitate the effect of a pump in the flow by using volume forces.

As we now calculate on a time cylinder $[0, T]$ with $T = 8$, we increase the number of timesteps on the coarsest time level to $N = 20$, leading to 40, 80, 160 and 320 timesteps on space-time level 2, 3, 4 and 5, respectively. The regularisation parameters are chosen as $\alpha = 0.01$ and $\gamma = 0.1$, the solver configuration is the same as in the previous section. Table

Nonl. Solv.	Δt	Space-Lv.	#NL	#MG
Newton	1/10	2	3	7
	1/20	3	3	9
	1/40	4	3	9
	1/80	5	3	8

Table 4.14: *Flow-around-cylinder* example with oscillating flow. Convergence behaviour of the solver.

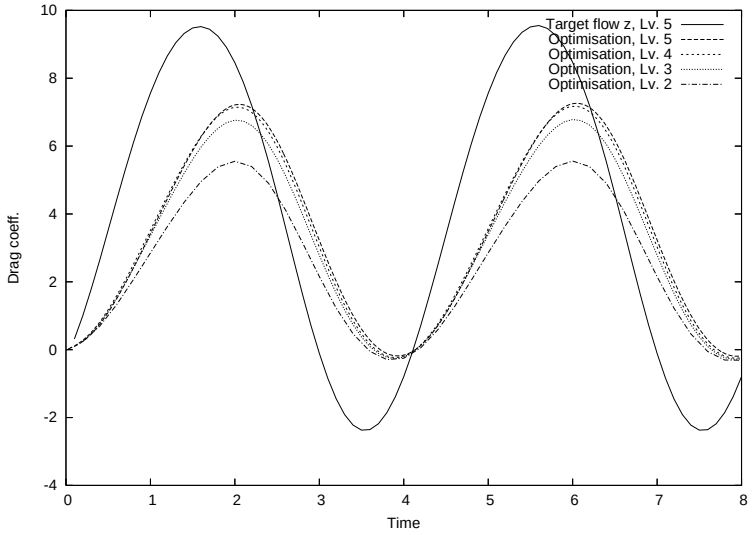

Figure 4.16: Drag coefficients in the *flow-around-cylinder* example with oscillating inflow boundary conditions. The solid black line shows the target solution obtained by simulation.

4.14 shows the convergence behaviour of the Newton optimisation solver. For all levels of refinement, the number of nonlinear iterations stays constant at a value of 3 and the number of linear iterations at 9, which means that we have about 3 multigrid iterations per nonlinear iteration.

For checking the solution quality, we measure the drag coefficients on the circular boundary component Γ_5. Figure 4.16 visualises the drag coefficients around the cylinder for the time interval under consideration. The black line belongs to the drag coefficients in the simulation while the dashed and dotted lines depict the results from optimisation at different space/time refinement levels. The maximum drag coefficient increases with increasing refinement level and has a relative error to the maximum reference drag of approx. 25%. The frequencies of both curves, those from the simulation and those from the optimisation, are identical, although the curve obtained by optimisation is slightly time-shifted by a time difference of $\Delta t \approx 0.5$. Nevertheless, the optimisation solver successfully controlled the fully nonstationary flow by volume force to imitate pulsating Dirichlet inflow boundary conditions.

5 Conclusions

Optimal control of the time-dependent Navier–Stokes equation can be carried out with iterative nonlinear and linear solution methods that act on the whole space-time cylinder. A nonlinear Newton approach allows fast convergence of the global solution. 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 space-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 $C \approx 10 - 30$. Being based on finite elements, the solver can be applied to rather general computational meshes.

This article concentrated on the basic ingredients for the nonlinear solver, namely the discretisation of the system for the Newton iteration, the discretisation of the global linear space time systems, the basic nonlinear iteration and the ingredients of the underlying space-time multigrid solver which is used for preconditioning. For simplicity, we restricted to first order implicit Euler discretisation in time and ignored any restrictions on the controls. Higher order schemes like Crank-Nicolson and bounds on the control will be focused on in a forthcoming publication. Algorithmic details about memory management, other possible preconditioners in space as well as space and time, basic approaches for optimisation as well as parallelisation will be the main focus of [15].

References

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

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

[3] A. Borzi. Multigrid methods for parabolic distributed optimal control problems. *J. Comput. Appl. Math.*, 157:365–382, 2003.

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

[5] 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`.

[6] 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.

[7] 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.

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

[9] W. Hackbusch. *Multi-Grid Methods and Applications*. Springer, Berlin, 1985.

[10] 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.

[11] M. Hinze, M. Köster, and S. Turek. A hierarchical space-time solver for distributed control of the Stokes equation. Preprint series DFG SPP1253, Deutsche Forschungsgesellschaft, 2008.

[12] 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.

[13] M. Hinze and S. Ziegenbalg. Optimal control of the phase interface during solidification of a gaas melt. Proceedings of the ICIAM 2007, Zürich, INRIA Sophia-Antipolis, Laboratoire Odyssee, 2007. to appear in PAMM (2008).

[14] 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`.

[15] M. Köster. *A Parallel High Performance Flow Solver for Optimisation with PDE Constraints*. PhD thesis, TU Dortmund, To appear 2009.

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

[17] 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`.

[18] S. Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. Springer, Berlin, 1999.

[19] 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. *CCPE*, 2008. Submitted.

[20] S. P. Vanka. Block-implicit multigrid solution of Navier-Stokes equations in primitive variables. *J. Comput. Phys.*, 65:138–158, 1986.

[21] H. Wobker and S. Turek. Numerical studies of Vanka-type smoothers in computational structural mechanics. *Adv. Appl. Math. Mech. (AAMM)*, 2008. Submitted.

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