Hamburger Beiträge
zur Angewandten Mathematik

A memory-reduced implementation of the Newton-CG method in optimal control of non-linear time dependent PDEs

Julia Sternberg and Michael Hinze

Nr. 2007-07
May 2007

A memory-reduced implementation of the Newton-CG method in optimal control of non-linear time dependent PDEs

Julia Sternberg
Department Mathematik,
University of Hamburg
D-20146 Hamburg, Germany,
email: sternberg@math.uni-hamburg.de

Michael Hinze
Department Mathematik,
University of Hamburg
D-20146 Hamburg, Germany,
email: hinze@math.uni-hamburg.de

Abstract

Derivative-based solution algorithms for optimal control problems of time dependent non-linear PDE systems require multiple solutions of backward-in-time adjoint systems. Since these adjoint systems in general depend on the primal state, and thus on forward information, the storage requirement for such solution algorithms is very large.

This paper proposes stable and memory efficient checkpointing techniques for evaluating gradients and Hessian times increment for such solution algorithms, and presents numerical tests with the instationary Navier-Stokes system which demonstrate that huge memory savings are achieved by the proposed approach while the increase in runtime is moderate. More precisely, a memory reduction of two orders of magnitude causes only a slow down factor of two in run-time.

Key words. Optimal control, Newton-CG approach, checkpointing, Navier-Stokes system

1 Introduction

To develop ideas let us consider the equation

$$G(y,u) = G_1(y) - Bu = 0 \qquad \text{in} \quad Z^* \tag{1}$$

as abstract realization of a time-dependent, nonlinear PDE, say here, $G : Y \times U \to Z^*$ denotes a sufficiently smooth mapping, and Y, U, and Z denote Hilbert spaces,

and $B : U \to Z^*$ a linear, bounded control operator. In the present paper we develop checkpointing techniques for derivative based solution algorithms for the following optimal control problem: *Find an optimal control* $u^* \in U$ *which minimizes the functional*

$$\hat{J}(u) := J(y(u), u) = J_1(y) + J_2(u), \tag{2}$$

where $y \in Y$ *and* u *are related through the equality constraints* (1).

To simplify the exposition we assume that $J = J_1(y) + J_2(u) : Y \times U \to \mathbb{R}$ and that J is sufficiently smooth. From now onwards we further assume that equation (1) for every $u \in U$ admits a unique solution $y(u)$ and that $G_y(y, u)$ admits for every $(y, u) \in Y \times U$ a bounded inverse.

As model algorithm for the numerical solution of problem (1) - (2) we consider the Newton-CG method. It iteratively determines the Newton direction by applying the conjugate gradient method to the Newton equation

$$\hat{J}''(u)\delta u = -\hat{J}'(u). \tag{3}$$

Here we assume that $\hat{J}''(u)$ is positive definite which certainly is satisfied in a small neighborhood of a nonsingular local minimum of $\hat{J}'(u)$. The whole optimization algorithm is then defined in terms of the following procedure:

Algorithm 1.1. (Newton-CG)
Given initial point u^0
For $k = 0, 1, 2, ...$
Compute the increment δu^k by applying the CG method to
$\hat{J}''(u^k)\delta u^k = -\hat{J}'(u^k)$, starting from $\delta u^k = 0$.
CG method
Set $i = 0$, $\delta u_0^k = 0$, $r_0 = \hat{J}''(u^k)\delta u_0^k + \hat{J}'(u^k)$, $p_0 = -r_0$.
While $\|r_i\| > TOL$
1. $\alpha_i = \frac{r_i^\top r_i}{p_i^\top \hat{J}''(u^k) p_i}$,
2. $\delta u_{i+1}^k = \delta u_i^k + \alpha_i p_i$,
3. $r_{i+1} = r_i + \alpha_i \hat{J}''(u^k) p_i$,
4. $\beta_{i+1} = \frac{r_{i+1}^\top r_{i+1}}{r_i^\top r_i}$,
5. $p_{i+1} = -r_{i+1} + \beta_{i+1} p_k$,
6. $i = i + 1$.
End CG method
Set $u^{k+1} = u^k + \delta u_i^k$.
End For

To perform Alg. 1.1 we first need expressions for the gradient $\hat{J}'(u)$ and the reduced Hessian $\hat{J}''(u)$ of the objective function $\hat{J}(u)$. Moreover, step 1. and step 3. of CG method require the evaluation of $\hat{J}''(u^k)p_i$ using the reduced Hessian $\hat{J}''(u^k)$ evaluated at the current Newton iterate u^k. Here, p_i denotes the conjugate direction.

Algorithms for computing the gradient $\hat{J}'(u)$ and the product of reduced Hessian times vector are specified in the following.

Introducing the *adjoint variable* $\lambda \in Z$ as a solution of the *adjoint equation*

$$G_y(y,u)^*\lambda = -J_y(y,u), \tag{4}$$

it is well known that the gradient of $\hat{J}$ can be expressed as

$$\hat{J}'(u) = J_u(y(u),u) + G_u(y(u),u)^*\lambda = J_{2_u}(u) + G_u(y(u),u)^*\lambda. \tag{5}$$

Calculation of the gradient $\hat{J}'(u)$ thus can then be summarized to the following procedure:

Algorithm 1.2. (Gradient)

1. Solve $G(y(u),u) = 0$ for $y \in Y$,
2. solve $G_y(y(u),u)^*\lambda = -J_y(y,u)$ for $\lambda \in Z$,
3. set $\hat{J}'(u) = J_u(y(u),u) + G_u(y(u),u)^*\lambda$.

In the present situation the reduced Hessian is given by

$$\begin{aligned}\hat{J}''(u) = G_u(y(u),u)^* G_y(y(u),u)^{-*} \{J_{yy}(y(u),u) + \\ \langle G_{yy}(y(u),u)(\cdot,\cdot),\lambda\rangle_{Z^*,Z}\} G_y(y(u),u)^{-1} G_u(y(u),u) + J_{uu}(y(u),u),\end{aligned} \tag{6}$$

where we have used $G_{yu} = G_{uy} = 0$. From its structure we conclude that the application of $\hat{J}''(u)$ to an element $\delta u \in U$ amounts to

Algorithm 1.3. (Reduced Hessian times vector)

1. Solve $G_y(y(u),u)v = G_u(y(u),u)\delta u$ for $v \in Y$,
2. form $rhs := J_{yy}(y(u),u)v + \langle G_{yy}(y(u),u)(v,\cdot),\lambda\rangle_{Z^*,Z}$,
3. solve $G_y(y(u),u)^*\mu = rhs$ for $\mu \in Z$,
4. evaluate $\tilde{\mu} := G_u(y(u),u)^*\mu$, and finally
5. set $\hat{J}''(u)\delta u = \tilde{\mu} + J_{uu}(y(u),u)\delta u = \tilde{\mu} + J_{2_{uu}}(u)\delta u$.

One observes, that the adjoint operator $G_y(y(u),u)^*$ comes into play to provide λ in step 2. of Alg. 1.2 and μ in step 3. of Alg. 1.3. Now, we think of (1) as the abstract realization of a time-dependent nonlinear PDE system on the time horizon $[0,T]$ with y denoting the state variables and u serving as control variables. Then, for given control u the computation of λ requires knowledge of the state $y(u)$ on the whole time horizon. Similar, the computation of μ in Alg. 1.3 requires knowledge

of the state variables y and v on the whole time horizon. In particular, for large time horizons with y and v representing a two- or three-dimensional quantities field storage may form a serious bottleneck. In the present paper we propose efficient checkpointing strategies to circumvent this storage problem.

Let us briefly comment on contributions to memory reduced computation of adjoints. Static and adaptive checkpointing techniques for the realization of Alg. 1.2 are developed in [3, 4, 11, 9, 16, 17, 15, 18]. Optimal checkpointing strategy is applied to the Burgers equation in [12]. The optimal static approach is applied to the Navier-Stokes system in [13], and in [14] for the same problem is extended to an adaptive memory and run-time reduced checkpointing strategy.

In the present paper we introduce checkpointing strategies for the memory-efficient implementation of Alg. 1.1. These strategies combine adaptive checkpointing of [9, 14] with static checkpointing for non-uniform step cost distributions as follows: An adaptive schedule is used to compute $\hat{J}'(u)$, which in turn induces a time discretization as well as a step cost distribution, which is kept fixed for performing subsequent static checkpointing in the CG-iteration. To evaluate Hessian-vector products in the CG method, i.e. to implement Alg. 1.3, efficient checkpointing strategies are developed in the present paper. The computer implementation is based on the routine a-revolve, combining static and adaptive checkpointing for uniform and non-uniform step cost. This algorithmical tool is an extension of the package revolve which A. Griewank and A. Walther developed in [4].

The evaluation of the Hessian-vector product, particularly the realization of Alg. 1.3, can be interpreted as the evaluation of tangents of adjoints and is equivalent to the implementation by Algorithmic Differentiation (AD). The forward simulation, i.e. the time integration of the equality constraints (1), is divided into a sequence of elementary steps or operations through the time horizon, which are evaluated successfully. Parallel to this, we perform step 1. of Alg. 1.3, provided values of the direction δu are available. This procedure can be interpreted as a forward propagation of tangents, i.e. the forward mode of AD. Then, we evaluate adjoint variables during the reverse propagation of gradients, i.e. the reverse mode of AD. Parallel to this procedure, also in the reverse mode, we perform step 3. of Alg. 1.3 as a forward propagation of tangents applied to the evaluation of adjoints. During this procedure the so called tangents of adjoints are evaluated. For more details concerning forward and reverse mode of AD see [3] (Chapter 3 and 4).

The remaining part of this paper is organized as follows. Section 2 introduces checkpointing techniques, particularly efficient reversal schedules. In Section 3 the instationary Navier-Stokes equations are adapted to the setting introduced in Section 1. Section 4 describes the implementation of efficient checkpointing strategies applied to the optimal control problem governed by the Navier-Stokes equations. We illustrate the capabilities of efficient reversal schedules w.r.t. memory reduction and run-time effort in the calculation of the tangents of adjoints. Finally, in Section 5 we present some conclusions.

2 Reversal schedules

In this section we describe reversal schedules based on checkpointing techniques and develop new static checkpointing strategies for step sequences with non-uniform step cost. They can be applied to numerical calculation of adjoints and tangents of adjoints. In this context we refer to (1) and (4) as the realization of a forward-in-time PDE and a backward-in-time adjoint PDE, respectively.

2.1 Tangents of adjoints, basic, and binomial approaches

The numerical calculation of tangents of adjoints is based on appropriate discretizations of forward and adjoint PDEs. For calculating an approximation of $y(u)$ and v one has to evaluate subfunctions Y_{j+1} and V_{j+1}, $0 \leq j < \ell$, resulting from the time discretization of the forward PDE. These subfunctions act on the states y^j and v^j to calculate the subsequent intermediate states y^{j+1} and v^{j+1} for $0 \leq j < \ell$ depending on a control action $\bar{u}^j$, i.e.,

$$y^{j+1} = Y_{j+1}(y^j, \bar{u}^j) \qquad \text{and} \qquad v^{j+1} = V_{j+1}(y^j, v^j, \bar{u}^j) \; . \tag{7}$$

We combine two subfunctions Y_{j+1} and V_{j+1} to the forward time step $F_{j+1} = (Y_{j+1}, V_{j+1})^\top$, $0 \leq j < \ell$. In order to compute tangents of adjoints the discretization of the adjoint PDE yields subfunctions $\bar{Y}_{j+1}$ and $\bar{V}_{j+1}$ for $\ell > j \geq 0$ with

$$\lambda^j = \bar{Y}_{j+1}(y^j, \bar{u}^j, \lambda^{j+1}) \qquad \text{and} \qquad \mu^j = \bar{V}_{j+1}(v^j, \bar{u}^j, \mu^{j+1}, y^j, \lambda^{j+1}) \; , \tag{8}$$

which are combined to the adjoint time step

$$\bar{F}_{j+1}(y^j, v^j, \bar{u}^j, \lambda^{j+1}, \mu^{j+1}) = \left(\bar{Y}_{j+1}(y^j, \bar{u}^j, \lambda^{j+1}), \bar{V}_{j+1}(v^j, \bar{u}^j, \mu^{j+1}, y^j, \lambda^{j+1})\right)^\top . \tag{9}$$

The evaluation of $\bar{F}_{j+1}$ may require some intermediate results calculated during the computation of y^{j+1} and v^{j+1} from the previous states y^j and v^j, respectively. Hence, it is supposed that for each $0 \leq j < \ell$, there exists a recording step

$$\hat{F}_{j+1}(y^j, v^j, \bar{u}^j) = \left(\hat{Y}_{j+1}(y^j, \bar{u}^j), \hat{V}_{j+1}(y^j, v^j, \bar{u}^j)\right)^\top , \tag{10}$$

which causes the recording of intermediate values required during the evaluation of the time step F_{j+1} onto a data structure called *tape*. Using the recording step $\hat{F}_{j+1}$ and adjoint time step $\bar{F}_{j+1}$ the basic way to compute tangents of adjoints reads as follows.

Algorithm 2.1. (Basic approach)
Recording: Set y and v to the initial values y_0 and v_0.
do $j = 0, \ell - 1$
Perform $(y^{j+1}, v^{j+1})^\top = \hat{F}_{j+1}(y^j, v^j, \bar{u}^j)$

end do
Reverse: Set λ and μ to the end values λ^ℓ and μ^ℓ
do $j = \ell - 1, 0, -1$
Perform $(\lambda^j, \mu^j)^\top = \bar{F}_{j+1}(y^j, v^j, \bar{u}^j, \lambda^{j+1}, \mu^{j+1})$
end do

The storage requirement of the basic approach is proportional to the number ℓ of time steps because intermediate data of ℓ time steps are stored during the recording steps. Thus, the memory requirement of the basic approach may become a problem if we consider real-world problems, for example computing adjoints of 3D flows. Therefore, due to their size, only a very limited number of intermediate states, called checkpoints, can be kept in memory. Applying a checkpointing technique, the required intermediate values are generated piecewise by restarting the evaluation repeatedly from the suitably placed checkpoints. Therefore, the calculation of tangents of adjoints can be performed based on a checkpointing strategy, even in such cases where the basic method fails due to excessive memory requirement (see e.g. [4, 5]). These checkpointing strategies, the so called reversal schedules, can be formalized as follows.

Definition 2.2 (Reversal Schedule S)**.** For an evolution of ℓ time steps and c available checkpoints a *reversal schedule* S initializes $j = 0$ and subsequently performs a sequence of basic actions

$$\begin{array}{lcl} A_m & \equiv & \text{Increment } j \text{ by } m \in \{1, \ldots, \ell - j - 1\},\ A_m = F_{j+1} \circ ... \circ F_{j+m} \\ D & \equiv & \text{Decrement } \ell \text{ by 1 if } j = \ell - 1,\ D = \hat{F}_\ell \circ \bar{F}_\ell \\ W_i & \equiv & \text{Copy state } j \text{ to checkpoint } i \in \{1, 2, \ldots, c\} \\ R_i & \equiv & \text{Reset state } j \text{ to checkpoint } i \end{array}$$

until j has been reduced to 0, i.e., the reversal is finished.

To derive optimal reversal schedules, i.e. schedules that minimize the overall evaluation time, one has to take into account following parameters:

1. the number ℓ of time steps to be reversed;
2. the number c of checkpoints that can be accommodated; and
3. the step cost: $t_j = TIME(F_j)$, $\hat{t}_j = TIME(\hat{F}_j)$, $\bar{t}_j = TIME(\bar{F}_j)$.

To classify available reversal schedules we introduce some notations, needed in the following. Step cost are called **uniform** if $t = t_j$ for all $1 \leq j \leq \ell$, and **non-uniform** if $t_i \neq t_j$ for some $i \neq j$. Reversal schedules for a given and adaptive determined number of time steps are called **static** and **adaptive** reversal schedules, respectively. According to [9] we denote by $S_{bin}(\ell, c)$ and $S_{opt}(\mathbf{t}_\ell, c)$ the **optimal** static reversal schedules for uniform and non-uniform step cost, respectively, where $\mathbf{t}_\ell =< t_1, ..., t_\ell >$ denotes a step cost distribution in the non-uniform case. Analogously, we denote by $S_{adapt}(\ell, c)$ and $S_{adapt}(\mathbf{t}_\ell, c)$ adaptive reversal schedules for the

uniform and non-uniform case, respectively (see [9, 14]). Optimal static reversal schedules $S_{bin}(\ell, c)$ for uniform step cost are called **binomial** reversal schedules. One such binomial reversal schedule is shown in Fig. 1. Here, time steps are plotted

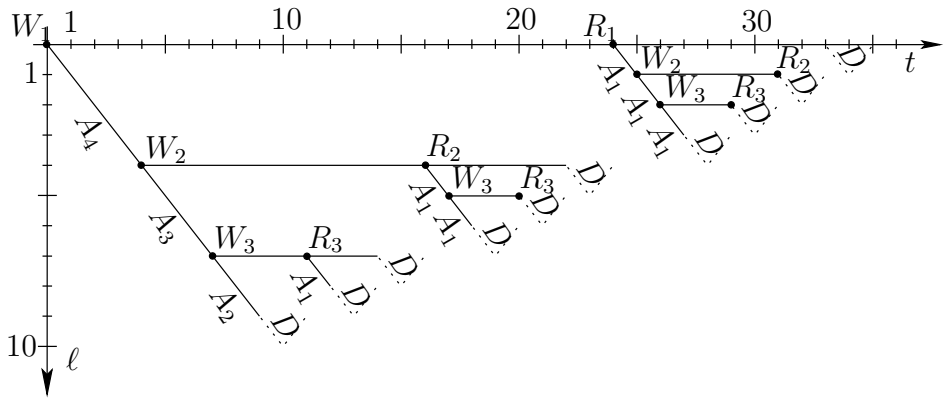

Figure 1: Binomial reversal schedules $S_{bin}(10, 3)$

along the vertical axis and computing time is represented by the horizontal axis. Hence, the horizontal axis can be thought of as the computational axis. Each solid horizontal line including the computational axis itself represents a checkpoint. The writing W_i and reading R_i of a checkpoint is marked with a dot. The solid slanted lines represent the actions A_m, i.e. the execution of time steps F_j without recording. The returning actions D are visualized by dotted slanted lines.

The explicit representations for the minimal evaluation cost $\mathbf{T}_{bin}(\ell, c)$ of binomial reversal schedules is

$$\mathbf{T}_{bin}(\ell, c) = r\ell - \beta(c+1, r-1), \tag{11}$$

with r being the unique integer satisfying

$$\frac{(c+r-1)!}{c!\,(r-1)!} = \beta(c, r-1) \;<\; \ell \;\leq\; \beta(c, r) = \frac{(c+r)!}{c!\,r!}. \tag{12}$$

For the derivation of this expression we refer to [3, 4]. The concept of binomial reversal schedules and its evaluation cost is crucial for the construction of efficient reversal schedules, which will be described in the following.

2.2 Efficient reversal schedules

According to [11], in the case of non-uniform step cost the complexity for determining an optimal reversal schedule $S_{opt}(\mathbf{t}_\ell, c)$ is $\mathcal{O}(\ell^2 c)$. Thus, its computation is extremely expensive, especially for large time horizons. For this situation we propose a heuristic reversal schedule, which is proved to be a very efficient, and performs very well in many numerical tests. In what follows let us denote static reversal schedules

constructed by this heuristic by $S_{ef}(\mathbf{t}_\ell, c)$, where ef stands for **efficient**. $\mathbf{T}_{ef}(\mathbf{t}_\ell, c)$ denotes the related evaluation cost of the reversal schedule $S_{ef}(\mathbf{t}_\ell, c)$. To construct efficient reversal schedules we first apply binomial reversal schedules to reverse step sequences with non-uniform step cost.

2.2.1 Upper bound for $\mathbf{T}_{bin}(\mathbf{t}_\ell, c)$

Consider the binomial reversal schedule $S_{bin}(\ell, c)$, applied for reversing a sequence of ℓ time steps with up to c checkpoints available and non-uniform step cost distribution $\mathbf{t}_\ell$. We denote the resulting evaluation cost $\mathbf{T}_{bin}(\mathbf{t}_\ell, c)$. Firstly, we construct an upper bound $\mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, c)$ for the evaluation cost $\mathbf{T}_{bin}(\mathbf{t}_\ell, c)$. Binomial reversal schedules are constructed without regarding temporal complexities of single time steps. For more or less homogeneous step cost distributions the application of such schedules may lead to acceptable results. But for step cost distributions with an essential difference in temporal complexities of single steps it can happen that the most expensive steps are evaluated most frequently, since a particular step cost distribution has no influence on the construction of a binomial reversal schedule. Examples for such situations are discussed in [9]. To avoid the high costs of reversing such step sequences the application of schedules constructed without regarding temporal complexities of single steps is not advisable.

Nevertheless, although binomial reversal schedule $S_{bin}(\ell, c)$ can not be accepted as an efficient reversal schedule $S_{ef}(\mathbf{t}_\ell, c)$ for variable step cost distributions, we can use an upper bound for its evaluation cost as an upper bound for the cost $\mathbf{T}_{ef}(\mathbf{t}_\ell, c)$. The evaluation cost $\mathbf{T}_{bin}(\mathbf{t}_\ell, c)$ is calculated in the following way:

$$\mathbf{T}_{bin}(\mathbf{t}_\ell, c) = \sum_{i=1}^{\ell} r_i \cdot t_i, \tag{13}$$

where r_i denotes how many times a time step F_i is evaluated during the execution of the binomial reversal schedule $S_{bin}(\ell, c)$. We refer to r_i as a **repetition number** of F_i.

There is no explicit formula for calculating a repetition number r_i for each time step F_i, $1 \leq i \leq \ell$. If one is interested in this task, the best way to proceed would be to follow the application of the binomial reversal schedule $S_{bin}(\ell, c)$ step by step. This obviously would result in additional costs. However, we are more interested in the upper bound $\mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, c)$, than in the exact value for $\mathbf{T}_{bin}(\mathbf{t}_\ell, c)$. Therefore, we evaluate values of maximal repetition numbers $r(c, \check{\ell})$ and $r(c-1, \ell-\check{\ell})$ for sequences $(F_1, ..., F_{\check{\ell}})$ and $(F_{\check{\ell}+1}, ..., F_\ell)$, respectively, with $\check{\ell}$ being the number of the intermediate state stored into the second checkpoint. Furthermore, on most numerical examples the repetition numbers r_i in most steps take one of the values of $r(c, \check{\ell})$ or $r(c-1, \ell-\check{\ell})$, respectively, or their difference to these values is small.

The value of $\mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, c)$ then is evaluated as

$$\mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, c) = \sum_{i=1}^{\ell} \widehat{r}_i \cdot t_i, \tag{14}$$

where the quantities $\widehat{r}_i$, $1 \leq i \leq \ell$, are specified in Table 1. Let us explain the construction of $\mathbf{G}^{\check{\ell}}_{bin}$ in terms of an example.

Example 2.3. Consider a sequence of 30 time steps with a non-uniform step cost distribution $\mathbf{t}_\ell = \langle t_1, ..., t_{30} \rangle = \langle 1, 2, ..., 29, 30 \rangle$. Let the number of available checkpoints be three. In Fig. 2 we sketch, which values $\widehat{r}_i$, $1 \leq i \leq 30$, are taken for evaluating the upper bound $\mathbf{G}^{16}_{bin}(\mathbf{t_{30}}, 3)$.

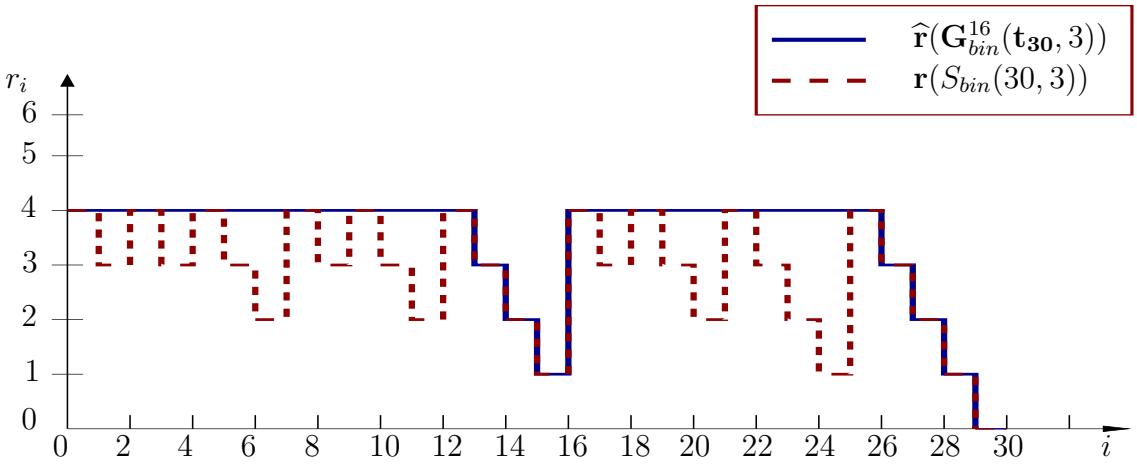

Figure 2: Repetition number profiles $\mathbf{r}(S_{bin}(30, 3))$ and $\widehat{\mathbf{r}}(\mathbf{G}^{16}_{bin}(\mathbf{t_{30}}, 3))$

Data of the 16th intermediate state is stored in the second checkpoint in this example. The values $r(c, \check{\ell})$ and $r(c-1, \ell-\check{\ell})$ for maximal repetition numbers are evaluated as follows:

$$r(c, \check{\ell}) = r(3, 16) = 3 \quad \textit{from} \quad \beta(3, r(3, 16) - 1) < 16 \leqslant \beta(3, r(3, 16)), \tag{15}$$
$$r(c-1, \ell-\check{\ell}) = r(2, 14) = 4 \quad \textit{from} \quad \beta(2, r(2, 14) - 1) < 14 \leqslant \beta(2, r(2, 14)). \tag{16}$$

The following values are assigned to the quantities $\widehat{r}_i$:

$$\begin{aligned}
&\widehat{r}_i = r(3, 16) + 1, && i = 1, ..., 16 - r(3, 16),\\
&\widehat{r}_i = 17 - i, && i = 17 - r(3, 16), ..., 16,\\
&\widehat{r}_i = r(2, 14), && i = 17, ..., 30 - r(2, 14),\\
&\widehat{r}_i = 30 - i, && i = 31 - r(2, 14), ..., 30.
\end{aligned}$$

□

$\widehat{r}_i = r(c,\check{\ell}) + 1,$	$i = 1, ..., \check{\ell} - r(c,\check{\ell})$
$\widehat{r}_i = \check{\ell} - i + 1,$	$i = \check{\ell} - r(c,\check{\ell}) + 1, ..., \check{\ell}$
$\widehat{r}_i = r(c-1, \ell - \check{\ell}),$	$i = \check{\ell} + 1, ..., \ell - r(c-1, \ell - \check{\ell})$
$\widehat{r}_i = \ell - i,$	$i = \ell - r(c-1, \ell - \check{\ell}) + 1, ..., \ell$

Table 1: Evaluation of $\widehat{\mathbf{r}}(\mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, c)) = \langle \widehat{r}_1, ..., \widehat{r}_\ell \rangle$

The computation of $r(c,\check{\ell})$, $r(c-1,\ell-\check{\ell})$, $\widehat{r}_i$ and of the upper bound $\mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, c)$ can be generalized for any arbitrary values of ℓ and c. First, $r(c,\check{\ell})$ and $r(c-1,\ell-\check{\ell})$ are computed by the relations:

$$r(c,\check{\ell}) \quad from \quad \beta(c, r(c,\check{\ell}) - 1) < \check{\ell} \leqslant \beta(c, r(c,\check{\ell})), \quad \text{and} \tag{17}$$

$$r(c-1,\ell-\check{\ell}) \quad from \quad \beta(c-1, r(c-1,\ell-\check{\ell}) - 1) < \ell - \check{\ell} \leqslant \beta(c-1, r(c-1,\ell-\check{\ell})), \tag{18}$$

where $\beta(c,r) = \frac{(c+r)!}{c!\, r!}$. The computation of values $r(c,\check{\ell})$ and $r(c-1,\ell-\check{\ell})$ can be very effectively arranged by iteration. With $r(c,\check{\ell})$ and $r(c-1,\ell-\check{\ell})$ available, the values of $\widehat{r}_i,\ i = 1, ...\ell$, are obtained according to Table 1. Therefore, the relationship (14) for the evaluation of the upper bound $\mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, c)$ can be represented in the following way:

$$\begin{aligned} \mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, c) = \sum_{i=1}^{\ell} \widehat{r}_i \cdot t_i = (r(c,\check{\ell}) + 1) \cdot \sum_{i=1}^{\check{\ell} - r(c,\check{\ell})} t_i + \sum_{i=\check{\ell}-r(c,\check{\ell})+1}^{\check{\ell}} (\check{\ell} - i + 1) \cdot t_i + \\ + r(c-1, \ell - \check{\ell}) \cdot \sum_{i=\check{\ell}+1}^{\ell - r(c-1,\ell-\check{\ell})} t_i + \sum_{i=\ell-r(c-1,\ell-\check{\ell})+1}^{\ell} (\ell - i) \cdot t_i, \end{aligned} \tag{19}$$

where $\check{\ell} = \check{\ell}(\ell, c)$ is a fixed value for given ℓ and c (see [4, 11]).

Discrepancies between the upper bound $\mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, c)$ and the evaluation cost $\mathbf{T}_{bin}(\mathbf{t}_\ell, c)$ are illustrated in Fig. 3 for a set of step sequences of length $\ell, 0 \leq \ell < 300$ with step cost distribution $\mathbf{t}_\ell = \langle t_1, ..., t_\ell \rangle$. The latter is illustrated for $\ell = 300$ in Fig. 3, left. For $c = 4$ the evaluation costs $\mathbf{T}_{bin}(\mathbf{t}_\ell, 4)$ and the upper bound $\mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, 4)$ are depicted in Fig. 3, right.

2.2.2 Construction of efficient reversal schedules

To construct efficient reversal schedules we proceed in a similar way as for constructing binomial reversal schedules (see [4, 11]): determine a position for the second checkpoint, store a corresponding intermediate state into the second checkpoint, and consider the resulting time step subsequences separately.

For this purpose, the principle of computing the upper bound $\mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, c)$ is used. An upper bound $\mathbf{G}_{ef}(\mathbf{t}_\ell, c)$ for the evaluation cost $\mathbf{T}_{ef}(\mathbf{t}_\ell, c)$ is computed in the

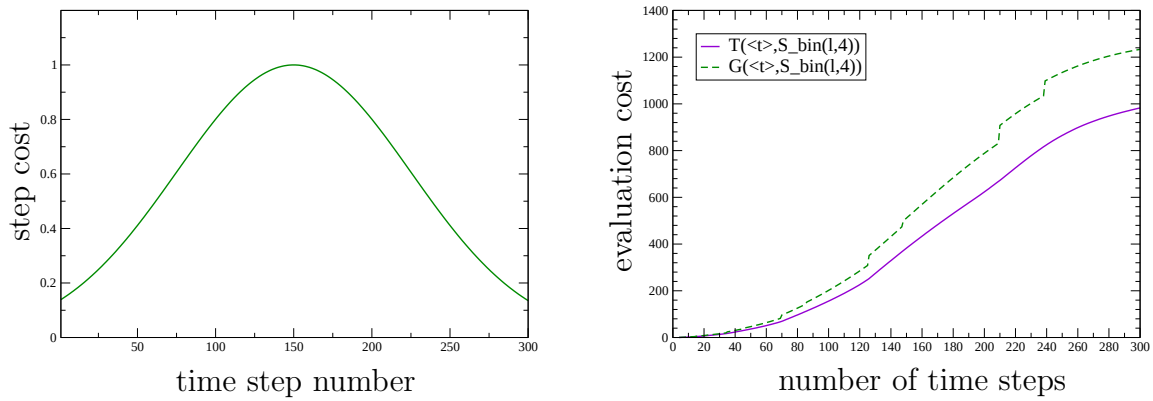

Figure 3: Left: Step cost distribution $\mathbf{t_{300}}$, right: Binomial evaluation cost $\mathbf{T}_{bin}(\mathbf{t}_\ell, 4)$ and its upper bound $\mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, 4)$

following way. Suppose, that $\check{\ell}$ is a position where the second checkpoint is stored. Then the corresponding upper bound $\mathbf{G}^{\check{\ell}}_{ef}(\mathbf{t}_\ell, c)$ is evaluated by

$$\mathbf{G}^{\check{\ell}}_{ef}(\mathbf{t}_\ell, c) = \mathbf{G}^{\check{\ell}}_{bin}(\mathbf{t}_\ell, c) \qquad \text{with} \qquad 0 < \check{\ell} < \ell. \tag{20}$$

and

$$\mathbf{G}_{ef}(\mathbf{t}_\ell, c) := \min_{0<\check{\ell}<\ell} \{\mathbf{G}^{\check{\ell}}_{ef}(\mathbf{t}_\ell, c)\}. \tag{21}$$

The value of $\check{\ell}$ where this minimum is attained determines the position where the second checkpoint is placed. In this way we establish a rule to determine a number of an intermediate state, which data should be stored into the second checkpoint during the execution of the efficient reversal schedule $S_{ef}(\mathbf{t}_\ell, c)$. This concept is applied to the subsequences $(F_1, ..., F_{\check{\ell}})$ and $(F_{\check{\ell}+1}, ..., F_\ell)$ with c and $(c-1)$ available checkpoints, respectively. If all available checkpoints are occupied, the remaining steps are reversed one by one. The described method is applied recursively till all time steps $F_i,\ 1 \le i \le \ell$, are reversed. This approach is summarized in

Algorithm 2.4. (Efficient Reversal Schedules)

$\mathbf{t_{b_0+1,\ell_0}} = \langle t_{b_0+1}, ..., t_{\ell_0} \rangle$; $\mathbf{T}_{ef}(\mathbf{t_{b_0+1,\ell_0}}, c) = 0$; $b := b_0$; $e := \ell_0$;
efficient(b,e,c)
begin
  if $((e - b > c)$ and $(c > 1))$
    $k = \text{argmin}_{b<\check{\ell}<e}\{\mathbf{G}^{\check{\ell}}_{ef}(\mathbf{t_{b+1,e}}, c)\}$
    store kth intermediate state into the second checkpoint
    $\mathbf{T}_{ef}(\mathbf{t_{b_0+1,\ell_0}}, c) += \sum_{i=b+1}^{k} t_i$;
    efficient(b,k,c);
    efficient(k,e,c-1);

fi;
if $(e - b \leq c)$
$\mathbf{T}_{ef}(\mathbf{t}_{\mathbf{b_0+1},\ell_0}, c) += \sum_{i=b+1}^{e-1} t_i;$
fi;
if $(c = 1)$
$\mathbf{T}_{ef}(\mathbf{t}_{\mathbf{b_0+1},\ell_0}, c) += \sum_{j=b+1}^{e} (e - j) \cdot t_j;$
fi;
end

As can be seen in Alg. 2.4, in order to determine a place for the second checkpoint we have to compare certain values of $\mathbf{G}_{ef}^{\check{\ell}}$. However, in general these numbers are rapidly reduced in subsequent algorithmic steps, except for the extremal situation where all checkpoints have to be set each time at the very end of the step sequence. Concluding, the temporal complexity needed for construction of efficient reversal schedules $S_{ef}(\mathbf{t}_\ell, c)$ as proposed in Alg. 2.4, is in most cases negligible compared to that for the construction of optimal reversal schedules $S_{opt}(\mathbf{t}_\ell, c)$ in the non-uniform case.

The example in Fig. 4 shows an efficient reversal schedule $S_{ef}(\mathbf{t}_\ell, c)$ constructed using Alg. 2.4. Consider a sequence of seven time steps with a step cost distribution $\mathbf{t_7} = \langle t_1, ..., t_7 \rangle = \langle 1, 2, ..., 6, 7 \rangle$. Three checkpoints are available. The evaluation

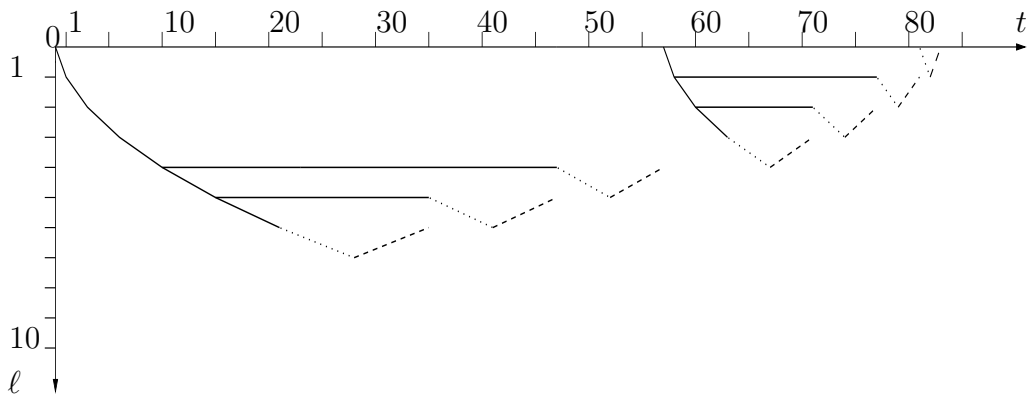

Figure 4: Efficient reversal schedule $S_{ef}(\mathbf{t_7}, 3)$

cost $\mathbf{T}_{ef}(\mathbf{t_7}, 3)$ needed for the execution of this reversal schedule amounts to 27 units. The effort $\mathbf{T}_{bin}(\mathbf{t_7}, 3)$, resulting from the reversal of this step sequence using the binomial reversal schedule $S_{bin}(7, 3)$, amounts to 30 units. Further numerical comparisons of efficient, optimal, and binomial reversal schedules are presented in [9] and in Section 4 of this paper, where efficient reversal schedules are applied to

implement the inexact Newton method for the optimal control problem governed by the Navier-Stokes equations.

2.2.3 Computer implementation of efficient reversal schedules

Alg. 2.4 is implemented in the computer routine a-revolve, involving static and adaptive checkpointing techniques. The corresponding checkpointing algorithm for implementing the Newton-CG method (Alg. 1.1) is defined in terms of the following procedure:

Algorithm 2.5. (Checkpointing Algorithm for Newton-CG)

snaps = c capo = 0 adaptive = 0 or 1 check = -1 turn = 0
if (adaptive = 0): fine = ℓ
else: fine = 0
do
  $t_{capo} = \Delta t_{capo}$
  whatodo = a-revolve(check,capo,fine,t_{capo},w)
  switch(whatodo)

    case advance: F_{capo}, $t = t + \Delta t_{capo}$
    case takeshot: store(y,ystor,check)
    case firsturn: $\hat{F}_{capo} \circ$ init(lam) $\circ \bar{F}_{capo}$, turn = 1
    case youturn: $\hat{F}_{capo} \circ \bar{F}_{capo}$
    case restore: restore(y,ystor,check)
    case error: print(schedule error!)

  end switch
  if (turn = 0) and (adapt = 1)
    fine = fine+1
while((whatodo $\neq$ terminate) && (whatodo $\neq$ error))

The first line of Alg. 2.5 contains initializations. Variable adapt = 1 if the number of time steps is unknown. Then, fine = 0 and we apply adaptive reversal schedules (for details see [9, 14]). This situation corresponds to the calculation of the gradient $\hat{J}'(u)$. Variable adapt = 0 if the number ℓ of time steps is known in advance. Then, fine = ℓ and efficient reversal schedules are applied to calculate the product reduced Hessian times vector which corresponds to $\hat{J}''(u)\delta u$. F_{capo}, $\hat{F}_{capo}$, and $\bar{F}_{capo}$ denote forward, recording, and adjoint steps, respectively, specified in Section 2.1.

The adjoint and the tangent of adjoint calculation is performed within a do-while-loop. Each execution of the loop-body starts with a call of a-revolve. Here, capo and fine determine the time steps to be reversed actually. The variables snaps and check denote the number of available and the number of actually stored checkpoints, respectively. t_{capo} denotes the cost of the forward time step F_{capo}. In our example t_{capo} is set to be equal to the time step size, $t_{capo} = \Delta t_{capo}$. By w an array is denoted,

which stores costs of all time steps. The value of turn shows whether the reversal sweep has been initialized or not.

Depending on the actual state of the reversal process, a-revolve returns a basic action according to Definition 2.2 of reversal schedules: advance $= A_1$, takeshot $= W_{check}$, restore $= R_{check}$, firsturn or youturn $= D$. As can be seen, the do-while-loop is completely independent of the actual problem to be reversed. The functions F_{capo}, $\hat{F}_{capo}$, and $\bar{F}_{capo}$ are required also for the basic approach to calculate adjoints and tangents of adjoints. Hence, in order to apply a reversal schedule to reduce the memory requirement, one has to code only the routines for storing and retrieving a checkpoint in addition to the already written software. Therefore, it is usually no problem to incorporate a reversal schedule into the calculation.

To implement a single Newton step in Alg. 1.1 we have to apply the Checkpointing Alg. 2.5 at least twice. Firstly, initializing adapt $= 1$ and utilizing adaptive reversal schedules to evaluate adjoints of the state equations in order to determine the value of the gradient $\hat{J}'(u)$, the number of time steps, and the step cost distribution, needed for the subsequent CG approach. Secondly, initializing adapt $= 0$ and utilizing static efficient reversal schedules to evaluate adjoints of the linearized state equations in order to calculate the product reduced Hessian times vector $\hat{J}''(u)\delta u$. The last scheme has to be applied for each conjugate direction and each increment δu within a single CG step.

3 Application to the Navier-Stokes system

As model application we now illustrate how derivatives of the reduced functional $\hat{J}$ in control of the instationary Navier-Stokes equations can be numerically realized utilizing the formalism developed in the previous sections. Since the focus of the present paper is a purely algorithmical one we reduce the presentation of functional analytic prerequisites to a necessary minimum. Interested readers are referred to [7, 8, 13]. Let us introduce the solenoidal spaces

$$H := \{v \in L^2(\Omega)^2, \text{div } v = 0\}^{clos_{L^2}} \text{ and } V := \{v \in L^2(\Omega)^2, \text{div } v = 0\}^{clos_{H^1}}.$$

From here onwards it is convenient to formulate the controlled instationary Navier-Stokes system in its primitive setting: given a control $u \in U$, find a solenoidal state y together with a pressure p such that

$$\left.\begin{array}{rcl} y_t - \nu\Delta y + (y \cdot \nabla)y + \nabla p &=& Bu \text{ in } Q, \\ -\text{div } y &=& 0 \text{ in } Q, \\ y(x,t) &=& 0 \text{ on } \partial\Omega \times (0,T), \\ y(x,0) &=& y_0(x) \text{ in } \Omega \end{array}\right\} \tag{22}$$

is satisfied, where $y_0 \in H$ denotes the initial value. Here, $\Omega \subset \mathbb{R}^2$ denotes the spatial domain, $Q := (0,T)\times\Omega$ is the time-space cylinder and B denotes the control operator

which maps elements of the abstract Hilbert space U of controls to admissible right-hand sides. In the formalism of Section 1 this system is represented in the form of the operator equation $G(y,u) = 0$, where $G : Y \times U \to Z^* := L^2(V^*) \times H$ and $Y := \{v \in L^2(V), v_t \in L^2(V^*)\}$. For details and notation we refer to the book [10] of Temam.

In order to express the actions of $\hat{J}'(u)$ and $\hat{J}''(u)$ for the cost functional given in equation (2) we have to provide those of $G_y^{-1}(y,u)$ and $G_y^{-*}(y,u)$, respectively. To describe these actions let for given $J_y(y,u) \in Y^*$ the vector function $\lambda \in Z$ be defined by

$$\lambda \;=\; (\lambda^1, \lambda^2) \;=\; -G_y(y,u)^{-*} J_y(y,u), \tag{23}$$

and let for $(f, v_0) \in Z$

$$v := G_y(y,u)^{-1}(f, v_0). \tag{24}$$

It is shown in [7, 8] that under suitable regularity assumptions on J the adjoint variable λ of (23) together with the adjoint pressure ξ satisfies the system

$$\left.\begin{aligned} -\lambda_t^1 - \nu\Delta\lambda^1 - (y\cdot\nabla)\lambda^1 + (\nabla y)^t\lambda^1 + \nabla\xi &= -J_{1_y}^{(t)}(y) \text{ in } Q,\\ -\text{div } \lambda^1 &= 0 \text{ in } Q,\\ \lambda^1(x,t) &= 0 \text{ on } \partial\Omega\times(0,T),\\ \lambda^1(x,T) &= -J_{1_y}^{(T)}(y) \text{ in } \Omega, \end{aligned}\right\} \tag{25}$$

and $\lambda^2 = \lambda^1(0)$. The superscripts $(t), (T)$ refer to a possible dependence of the functional J_{1_y} on the time instances (t) and (T), respectively. Finally, together with some pressure ρ the function v of (24) satisfies

$$\left.\begin{aligned} v_t - \nu\Delta v + (y\cdot\nabla)v + (v\cdot\nabla)y + \nabla\rho &= f \text{ in } Q,\\ -\text{div } v &= 0 \text{ in } Q,\\ v(x,t) &= 0 \text{ on } \partial\Omega\times(0,T),\\ v(x,0) &= v_0(x) \text{ in } \Omega. \end{aligned}\right\} \tag{26}$$

Now, let $\lambda \equiv \lambda_1$. In order to compute approximations of (y,p), (λ,ξ), and (v,ρ) the partial differential equations have to be discretized appropriately. For the numerical tests presented in the subsequent section, Taylor-Hood finite elements are used for spatial discretization, i.e. continuous, piecewise quadratic polynomials for the velocity approximation and continuous, piecewise linear polynomials for the pressure approximation. As time discretization scheme for the Navier-Stokes system (22) we apply the semi-implicit Euler-method which performs implicit time stepping w.r.t. diffusive terms, while the convective terms are treated explicitly. The time integration is performed adaptively by the rule

$$\Delta t_j = 0.7\frac{h}{\max_{x\in\Omega} \mid y(t_j) \mid}, \tag{27}$$

where h denotes the grid size of the spatial discretization. We note that $y(t_j) \in \mathcal{C}(\bar{\Omega})$ can be guaranteed for $t_j \in (0,T)$ if $Bu \in L^2(Q)^2$ and $y_0 \in V \cap H^2(\Omega)^2$. These

conditions form the minimal regularity requirements for proving error estimates for numerical approximation schemes of the Navier-Stokes system, compare [2, 6]. Since Δt_j depends on the flow field $y(t_j)$ it is only possible to evaluate Δt_j for the current time step. Therefore, the number l of time steps is not known in advance. The resulting numerical scheme for a state $y^j \sim y(t_j)$ with $0 \leq j < \ell$ is given by

$$\begin{aligned} \frac{y^{j+1} - y^j}{\Delta t_j} - \nu \Delta y^{j+1} + \nabla p^{j+1} &= (Bu)^j - (y^j \nabla y^j) \text{ in } \Omega \\ -\operatorname{div} y^{j+1} &= 0 \text{ in } \Omega,\ y^{j+1} = 0 \text{ in } \partial\Omega \end{aligned} \tag{28}$$

with $y^0 = y(0)$. In the setting of Section 2 this forward integration scheme may be rewritten as

$$y^{j+1} = Y_{j+1}(y^j, \bar{u}^j), \tag{29}$$

where the time step function Y_{j+1} is given by $Y_{j+1}(y^j, \bar{u}^j) = Y(\Delta t_j, y^j, (Bu)^j)$ with

$$Y(\Delta t, y, z) := \Delta t \left(P - \Delta t \nu S\right)^{-1} \left(z - (y\nabla)\, y\right) + Py.$$

Here, S denotes the Stokes Operator and P defines the Leray projection $L^2(\Omega)^2 \to H$, see [1].

The time discretization scheme of the adjoint variables λ in (25) is more involved. Here we take the transpose of the semi-implicit time discretization of eqs. (26) given next; for $0 \leq j < \ell$ let

$$\begin{aligned} \frac{v^{j+1}}{\Delta t_j} - \nu \Delta v^{j+1} + \nabla \rho^{j+1} &= f^j + \frac{v^j}{\Delta t_j} - (y^j \nabla v^j) - (v^j \nabla y^j) \text{ in } \Omega \\ -\operatorname{div} v^{j+1} &= 0 \text{ in } \Omega,\ v^{j+1} = 0 \text{ on } \partial\Omega, \end{aligned} \tag{30}$$

where $v^0 = v(0)$. To derive the time integration scheme for the adjoint variables it is convenient to reformulate the initial condition for v in the form

$$\begin{aligned} \frac{v^0}{\Delta t_0} - \nu \Delta v^0 + \nabla \rho^0 &= \frac{v_0}{\Delta t_0} - \nu \Delta v_0 \text{ in } \Omega \\ -\operatorname{div} v^0 &= 0 \text{ in } \Omega,\ v^0 = 0 \text{ on } \partial\Omega, \end{aligned} \tag{31}$$

where we set $\rho^0 \equiv 0$ and require $v_0 \in V \cap H^2(\Omega)^2$.

Formally transposing the scheme (30), (31) we obtain for the adjoint equations and $j = \ell - k \geq 0$ the integration scheme

$$\begin{aligned} \frac{\lambda^{\ell-k}}{\Delta t_{\ell-k}} - \nu \Delta \lambda^{\ell-k} &= \frac{\lambda^{\ell-k+1}}{\Delta t_{\ell-k}} - \nabla \xi^{\ell-k} - J_{1_y}^{(t_{\ell-k})}(y^{\ell-k}) \\ &\quad + (\lambda^{\ell-k+1} \nabla y^{\ell-k}) - (\nabla y^{\ell-k})^t \lambda^{\ell-k+1} \text{ in } \Omega, \\ -\operatorname{div} \lambda^{\ell-k} &= 0 \text{ in } \Omega,\ \lambda^{\ell-k} = 0 \text{ on } \partial\Omega, \end{aligned} \tag{32}$$

where the states $y^{\ell-k}$ are given by the solution of (28). We note, that the final state λ^ℓ in this integration scheme satisfies the quasi-Stokes system

$$\begin{aligned}\frac{\lambda^\ell}{\Delta t_\ell} - \nu\Delta\lambda^\ell - \nabla\xi^\ell = & -J_{1_y}^{(t_\ell)}(y^\ell, u^\ell) \text{ in } \Omega,\\ -\text{div } \lambda^\ell = 0 \text{ in } \Omega,\ \lambda^\ell = & \ 0 \text{ on } \partial\Omega.\end{aligned}$$

The adjoint integration scheme may be rewritten as

$$\lambda^j = \bar{Y}_{j+1}(y^j, \bar{u}^j, \lambda^{j+1}) = \bar{Y}(\Delta t_j, y^j, \lambda^{j+1}), \tag{33}$$

where

$$\bar{Y}(\Delta t, y, z) := \Delta t\,(P - \nu\Delta t S)^{-1}\left(-J_{1_y}^{(t)}(y) + (z\nabla)\,y - (\nabla y)^t\,z\right) + Pz. \tag{34}$$

Furthermore, one finds that the recording step $\hat{Y}(y^j, (Bu)^j)$ consists of the evaluation of $Y(\Delta t_j, y^j, (Bu)^j)$ and the storage of the new state vector y^{j+1}.

Next, let us discuss the time discretization of $\hat{J}''(u)\delta u$ described in Alg. 1.3. To begin with we note that

$$\langle G_{yy}(y(u), u)(a, b), (\lambda, \lambda^2)\rangle_{Z^*,Z} = \langle (a\nabla)b + (b\nabla a), \lambda\rangle_{L^2(V^*),L^2(V)},$$

which is independent of y and u, and $G_u(y, u) = (-B, 0)$. To discretize step 1. of Alg. 1.3 we propose to apply scheme (30) for the computation of v with $v_0 \equiv 0$, and y taken from (28). We recall that the time grid obtained from (27) is also used in this scheme, but now the number of time steps and the step length Δt_j are fixed. In the setting of Section 2 this forward integration scheme may be rewritten as

$$v^{j+1} \ = \ V_{j+1}(y^j, v^j, \bar{u}^j) = V(\Delta t_j, y^j, v^j, (Bu)^j), \tag{35}$$

with

$$V(\Delta t, y, v, z) := \Delta t\,(P - \Delta t\nu S)^{-1}\,(f(z) - (y\nabla)\,v - (v\nabla)\,y) + Pv.$$

As can be seen, equations (29) and (35) exactly match the time stepping (7) which forms the basis for the reversal schedules presented in Section 2.

In step 2. of Alg. 1.3 the numerical results of schemes (28) for y and (32) for λ are utilized to evaluate rhs. Finally, for the numerical computation of μ in step 3. we propose to use again scheme (32), where the terms containing the functional J have to be replaced by the corresponding terms of rhs from step 2. We emphasize that in general both, the state y and the variable v enter into rhs. This adjoint integration scheme may be rewritten as

$$\mu^j = \bar{V}_{j+1}(v^j, \bar{u}^j, \mu^{j+1}, y^j, \lambda^{j+1}) = \bar{V}(\Delta t_j, v^j, \mu^{j+1}, y^j, \lambda^{j+1}), \tag{36}$$

where

$$\bar{V}(\Delta t, v, \mu, y, \mu) := \Delta t \left(P - \nu \Delta t S\right)^{-1} \left(rhs(y, \lambda) + (\mu \nabla)\, v - (\nabla v)^t\, \mu\right) + P\mu. \quad (37)$$

Then, (33) and (36) exactly match (8), e.g. the adjoint time steps exactly fit for the application of the reversal schedules. Furthermore, one finds that the recording step $\hat{V}_{j+1}(y^j, v^j, \bar{u}^j)$ consists of the evaluation of $V(\Delta t_j, y^j, v^j, (Bu)^j)$ and the storage of the new state vector v^{j+1}.

4 Numerical results

A cavity flow problem serves as numerical example. The domain is defined by the unit square $\Omega := (0,1) \times (0,1)$. The final time is normalized to one, i.e., $T = 1$, and $\nu = 1/\mathrm{Re}$ with Re= 10. The vector

$$y_0(x) \;=\; e \begin{bmatrix} (\cos 2\pi x_1 - 1) \sin 2\pi x_2 \\ -(\cos 2\pi x_2 - 1) \sin 2\pi x_1 \end{bmatrix}$$

with e denoting the Euler number is used as initial condition. The control goal consists in approximating the time-dependent desired state given by

$$z(x,t) \;=\; \begin{bmatrix} \varphi_{x_2}(x_1, x_2, t) \\ -\varphi_{x_1}(x_1, x_2, t) \end{bmatrix}.$$

Here, φ is defined through the stream function

$$\varphi(x_1, x_2, t) = \theta(x_1, t)\theta(x_2, t) \quad \text{with} \quad \theta(y, t) = (1-y)^2(1 - \cos 2k\pi t), \quad y \in [0,1].$$

In order to measure the quality of the approximation the tracking-type cost function

$$\hat{J}(u) = J(y,u) \;=\; \frac{1}{2}\int_0^T \int_\Omega |\, y - z \,|^2 \, dx\, dt + \frac{c}{2}\int_0^T \int_\Omega |\, u \,|^2 \, dx\, dt \qquad (38)$$

with $c = 0.01$ is chosen. Fig. 5 shows the cavity flow at $t = 0.01$ together with the desired flow at $T = 1$.

To solve this optimal control problem we apply Newton-CG method described in Alg. 1.1. An implementation of this algorithm using the basic approach would require large memory amounts. Reversal schedules presented in Section 2 offer an opportunity to reduce this memory requirement drastically, while increasing the run-time only moderately. For this purpose, the routine a-revolve may be applied to compute the gradient $\hat{J}'(u)$ and the product of reduced Hessian times vector $\hat{J}''(u)\delta u$ (see Alg. 2.5).

In the remainder of this section we investigate the numerical performance of a-revolve. The application of adaptive reversal schedules for evaluating adjoints of Navier-Stokes systems is analyzed in [14]. Thus, in the sequel of this paper we focus

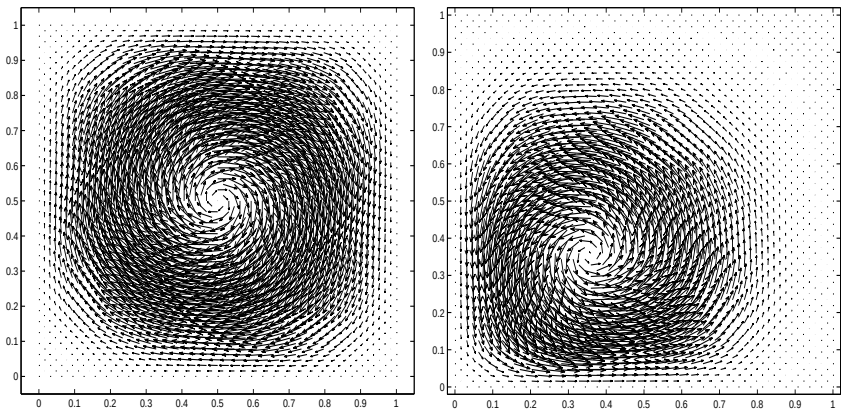

Figure 5: Left: Cavity flow at $t = 0.01$, right: Desired flow at $T = 1$

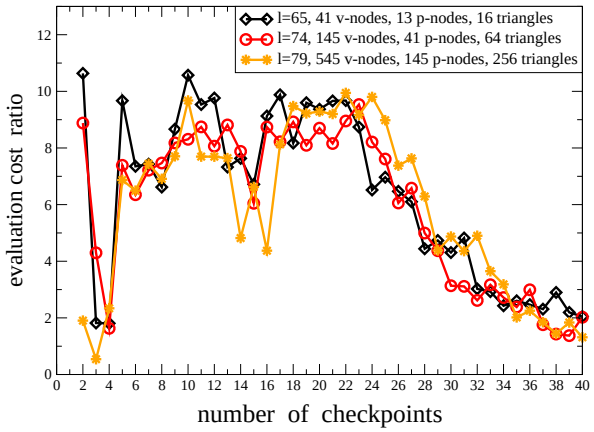

Figure 6: Evaluation cost needed to compute $\hat{J}''(u)\delta u$ using efficient and optimal reversal schedules

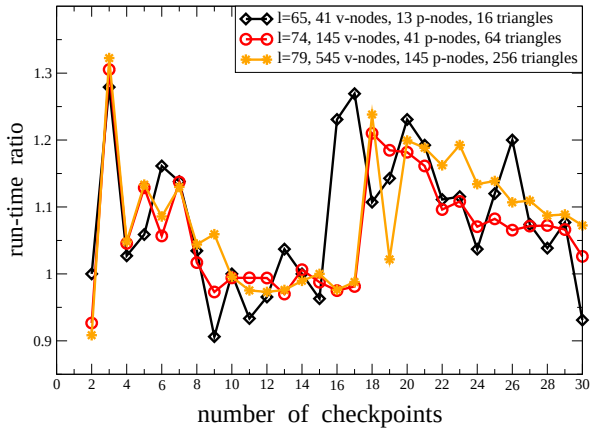

Figure 7: Run-time needed to compute $\hat{J}''(u)\delta u$ using efficient and optimal reversal schedules

on the application of efficient reversal schedules for evaluating the product $\hat{J}''(u)\delta u$ of reduced Hessian times vector.

Fig. 6 shows the ratio for the evaluation cost of efficient and optimal reversal schedules applied to compute $\hat{J}''(u)\delta u$. This ratio is given by

$$\frac{T(S_{ef}(\mathbf{t}_\ell, c)) - T(S_{opt}(\mathbf{t}_\ell, c))}{T(S_{opt}(\mathbf{t}_\ell, c))} * 100, \tag{39}$$

and presented on the vertical axis. As can be seen $T(S_{ef}(\mathbf{t}_\ell, c)) - T(S_{opt}(\mathbf{t}_\ell, c))$ varies between 5 and 12 % relative to the optimal evaluation cost $T(S_{opt}(\mathbf{t}_\ell, c))$ for $2 \leq c \leq 40$ checkpoints. The evaluation cost ratio only compares the algorithmical performance of efficient and optimal reversal schedules in terms of step costs. For practical implementations it is more reasonable also to investigate the resulting computational run-time, which is shown in Fig. 7.

The product $\hat{J}''(u)\delta u$ is computed applying a-revolve with various numbers of checkpoints. Fig. 7 shows the observed run-time behavior for different number of time steps. Here, the vertical axis gives the ratio of the run-time needed to compute $\hat{J}''(u)\delta u$ using efficient and optimal reversal schedules. The horizontal axis denotes the number of checkpoints used by the reversal schedule. As can be seen, the computational run-time resulting from the application of efficient reversal schedules compares to that needed by optimal reversal schedules, and sometimes is even smaller. This can be explained by the additional computational requirement for constructing an appropriate optimal reversal schedule.

Fig. 8 illustrates the ratio of run-time needed to compute $\hat{J}''(u)\delta u$ formed with the run-time to compute $\hat{J}(u)$. To compare the achieved results with the basic ap-

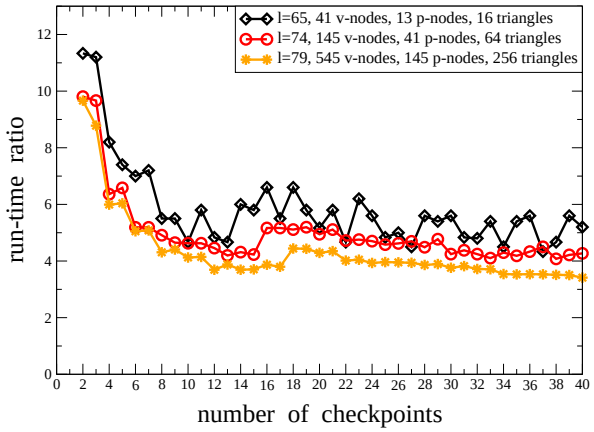

Figure 8: Run-time needed to compute $\hat{J}''(u)\delta u$ using efficient reversal schedules compared to that needed to compute the functional $\hat{J}(u)$

proach for computing tangents of adjoints, we note that the run-time for computing $\hat{J}''(u)\delta u$ is bounded by a small constant times the run-time to compute $\hat{J}(u)$. The value of the constant varies between seven and ten depending on the specific operation counts and memory accesses [3]. As can be seen, the run-time ratio for the checkpointing approach varies between four and seven for a reasonable number of checkpoints. This behavior results in a slow down factor up to two compared to the basic approach, where a complete trajectory is stored to compute the tangents of adjoints values. That is, using the checkpointing approach the computation of tangents of adjoints is at most twice as slow as the basic approach, where a complete forward trajectory is stored. Nevertheless, the memory requirement can be reduced enormously. Using a discretization with 2113 velocity nodes and 545 pressure nodes, one needs 76 kByte to store one checkpoint. Hence, if the reversal schedule utilizes six checkpoints, the memory requirement equals 456 kByte for calculating tangents of adjoints with the efficient reversal schedule. If the basic approach is applied, the memory requirement amount to 7.6 MByte. Therefore, efficient reversal schedules enable an immense memory reduction at a slight increase in run-time.

We observe that the dependence on the spatial discretization of all ratios presented so far is negligible. Fig. 8 therefore indicates that above a certain lower bound the number of checkpoints can be varied without having a considerable influence on the run-time behavior. This fact is also illustrated by the flat development of the run-time ratios in Fig. 8 if the number of checkpoints exceeds eight.

5 Conclusions

For adjoint and tangent of adjoint calculations one has to provide information computed during the forward integration in reverse order. The basic approach, namely the complete recording of the required information onto one stack, causes an enormous memory requirement. This paper presents efficient reversal schedules that allow a drastic reduction of the memory requirement while run-time compared to the basic approach increases only slightly. Moreover, efficient checkpointing causes only a slight increase in run-time compared to the optimal checkpointing. The resulting memory reduction and run-time behavior is studied for an optimal control problem based on incompressible Navier-Stokes equations by applying the checkpointing routine a-revolve. For this example it is shown that a memory reduction of two orders of magnitude causes only a slow down factor of two in run-time

References

[1] P. Constantin, C. Foias. *Navier-Stokes Equations*. The University of Chicago Press, 1988.

[2] K. Deckelnick, M. Hinze. Error estimates in space and time for tracking-type control of the instationary Stokes system. *ISNM*, 143:87–103, 2002.

[3] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, 2000.

[4] A. Griewank, A. Walther. Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Software*, 26:19 – 45, 2000.

[5] J. Grimm, L. Pottier, N. Rostaing-Schmidt. Optimal time and minimum space-time product for reversing a certain class of programs. In M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 95 – 106. SIAM Philadelphia, 1996.

[6] J.G. Heywood, R. Rannacher. Finite-element approximation of the nonstationary Navier-Stokes problem, I-IV. *SIAM J. Numer. Anal.*, 19:275-311,23:750-777,25:489-512,27:353-384, 1982-1990.

[7] M. Hinze. Optimal and instantaneous control of the instationary Navier-Stokes equations. Habilitationsschrift, 1999. Fachbereich Mathematik, Technische Universität Berlin.

[8] M. Hinze, K. Kunisch. Second order methods for optimal control of time-dependent fluid flow. *SIAM J. Control Optim.*, 40:925–946, 2001.

[9] J. Sternberg. Adaptive Umkehrschemata für Schrittfolgen mit nicht-uniformen Kosten. Diplomarbeit, 2002. Techn. Univ. Dresden.

[10] R. Temam *Navier-Stokes Equations*. North-Holland, 1979.

[11] A. Walther. *Program Reversal Schedules for Single- and Multi-processor Machines*. PhD thesis, Institute of Scientific Computing, TU Dresden, 1999.

[12] A. Walther, A. Griewank. Applying the checkpointing routine treeverse to discretizations of burgers' equation. In H.-J. Bungartz, F. Durst, and C. Zenger, editors, *High Performance Scientific and Engineering Computing*, volume 8 of *Lecture Notes in Computational Science and Engineering*, pages 13 – 24, Berlin/Heidelberg, 1999. Springer.

[13] M. Hinze, J. Sternberg, A. Walther. Discrete approximation schemes for reduced gradients and reduced Hessians in Navier-Stokes control utilizing an optimal memory-reduced procedure for calculating adjoints. Optimal Control Applications and Methods, 27:19-40, 2006

[14] M. Hinze, J. Sternberg. A-Revolve: An adaptive memory and run-time-reduced procedure for calculating adjoints; with an application to the instationary Navier-Stokes system. Optimization Methods and Software, 20(6):645–663, 2005.

[15] A. Walther, A. Griewank. Advantages of binomial checkpointing for memory-reduced adjoint calculations. In M. Feistauer et al., editors, Numerical mathematics and advanced applications, pages 834 – 843. Springer, 2004. Proceedings of ENUMATH 2003.

[16] M. Berggren. Numerical solution of a flow control problem: Vorticity reduction by dynamic boundary action. Siam J. Sci. Comput., 19(3):829–860, 1998.

[17] M. Berggren, R. Glowinski, JL. Lions. A computational approach to controllability issues for flow-related models. I: Pointwise control of the viscous Burgers equation. Int. J. Comput. Fluid Dyn. 7(3):237-252, 1996.

[18] V. Heuveline and A. Walther . Online checkpointing for parallel adjoint computation in PDEs: Application to goal oriented adaptivity and flow control. In W. Nagel et al., eds., Proceedings of Euro-Par 2006, LNCS 4128, pages 689 – 699, Springer, 2006.