A-Revolve: An adaptive memory-reduced procedure for calculating adjoints; with an application to computing adjoints of the instationary Navier-Stokes system

Michael Hinze
Institut für Numerische Mathematik,
Technische Universität Dresden
D–01069 Dresden, Germany,
email: hinze@math.tu-dresden.de

Julia Sternberg
Institut für Wissenschaftliches Rechnen,
Technische Universität Dresden
D–01069 Dresden, Germany,
email: jstern@math.tu-dresden.de

Contents

Abstract

The paper presents a low-storage and low-run-time approach for calculating numerical approximations of adjoint equations for the instationary Navier-Stokes equations with adaptive evaluation of the discretization step. It utilizes adaptive checkpointing. The amounts of memory reduction due to checkpointing and of the increase in run-time caused by repeated forward integration of the Navier-Stokes equations are reported. It is one result that memory reduction of two orders of magnitude only causes a slow down factor of 2-3 in run-time. It is a further result that the adaptive checkpointing for the instationary Navier-Stokes equations causes only a slight increase of the run-time compared to the static optimal checkpointing. The amounts of run-time increase caused by the adaptive checkpoint allocations compared with the static optimal checkpointing is reported as well.

Key words. Optimal control, Navier-Stokes system, adaptive checkpointing
AMS(MOS) subject classification. 49M15,49M29,65K10,76D55.

1 Introduction

Adjoints are the most important tool for computing sensitivities in control problems for the time dependent Navier-Stokes equations, in particular when the number of control variables is large and thus forbids approaches based on finite difference methods. To illustrate the role of adjoints for calculating sensitivities let us consider the following abstract optimal control problem: *Find an optimal control* $u^* \in U$ *which minimizes the functional*

$$\hat{J}(u) := J(y(u), u) = J_1(y) + J_2(u), \tag{1}$$

where $y \in Y$ *and* u *are related through the equality constraints*

$$G(y, u) = G_1(y) - Bu = 0. \tag{2}$$

Here we assume that $J = J_1(y) + J_2(u) : Y \times U \to \mathbb{R}$, $G : Y \times U \to Z^*$ are sufficiently smooth mappings, B denotes the linear, bounded control operator and $(U, (\cdot,\cdot)_U)$, $(Y, (\cdot,\cdot)_Y)$ and $(Z, (\cdot,\cdot)_Z)$ denote Hilbert spaces. For simplicity from now onwards we further assume that eq. (2) admits for every $u \in U$ a unique solution $y(u)$ and that $G_y(y, u)$ admits for every $(y, u) \in Y \times U$ a bounded inverse. In Section 3 we will show that control problems for the instationary Navier-Stokes equations fit into this setting.

For the derivatives of the functional $\hat{J}$ at point u in direction v we obtain, utilizing the chain rule

$$(\hat{J}'(u), v)_U = \langle J_y(y, u), y'(u)v\rangle_{Y^*,Y} + (J_u(y, u), v)_U.$$

The implicit function theorem applied to eq. (2) yields

$$y'(u) = -G_y(y, u)^{-1} G_u(y, u)$$

which implies

$$\langle J_y(y, u), y'(u)v\rangle_{Y^*,Y} = (-G_u(y, u)^* G_y(y, u)^{-*} J_y(y, u), v)_U.$$

Introducing the *adjoint variable* $\lambda := -G_y(y, u)^{-*} J_y(y, u) \in Z$ we finally can express the gradient of $\hat{J}$ as

$$\hat{J}'(u) = J_u(y(u), u) + G_u(y(u), u)^* \lambda. \tag{3}$$

In other words, all directional derivatives of the functional $\hat{J}$ can be calculated with the help of the adjoint variable λ, which in turn is obtained by one solve of the *adjoint equation*

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

By similar calculations and transformations we obtain the reduced Hessian, i.e. the second derivative, of the functional $\hat{J}$ (recall $J(y(u),u) = J_1(y) + J_2(u)$), which implies:

$$\begin{aligned}\hat{J}''(u) = G_u(y(u),u)^* G_y(y(u),u)^{-*} \{ J_{yy}(y(u),u) + \\ \langle G_{yy}(y(u),u)(\cdot,\cdot), \lambda\rangle_{Z^*,Z} \} \, G_y(y(u),u)^{-1} G_u(y(u),u) + J_{uu}(y(u),u),\end{aligned} \tag{5}$$

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

Algorithm 1.1. (Reduced Hessian times vector)

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

One observes, that the adjoint operator $G_y(y(u),u)^*$ comes into play: To provide μ in step 3. and also λ defined by eq. (4) for given $u \in U$. Let us note here that in the context of inexact Newton methods the evaluation of Hessian times increment is the basic building block.

Now we think of (2) as the abstract realization of the instationary Navier-Stokes equations 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 and μ requires knowledge of the state $y(u)$ on the whole time horizon. In particular for large time horizons with y representing a 2- or 3-dimensional flow velocity field storage forms a serious bottleneck.

As partial remedy to this storage problem in [10], an static optimal checkpointing procedure was developed and was applied to compute adjoints for the instationary Navier-Stokes equations. Its slow down factor in run-time caused by repeated forward integration of the Navier-Stokes equations only lies between 2 and 3 whereas storage reduction of two orders of magnitude is observed. The main algorithmical tool utilized was revolve which A. Griewank and A. Walther developed in [5].

However, optimal static checkpointing procedures strongly rely on knowledge of the number of time steps to be performed. In view of numerical simulations in practical applications this is a serious shortcoming of the method, since in most numerical simulations adaptive time-stepping is utilized and neither the number of iterate steps therefore is known in advance nor time grid and step cost. As remedy for these shortcomings in the present paper adaptive reversal schedules are introduced. These schedules propose and adaptively modify checkpointing distributions during the run-time of the forward solve. With this method the static concept realized in revolve is extended, but it cannot yet be claimed that the resulting checkpointing is optimal. However numerical investigations show that adaptive reversal schedules are good approximations of the related optimal static one. The numerical computation of adjoints for the instationary Navier-Stokes equations serves as model problem. The time integration of the Navier-Stokes equations is performed adaptively in order to simulate a typical scenario for the application of adaptive reversal schedules. It is one result of this work that the run-time ratio for the adaptive schedules is always very close to that of the optimal static schedules (see Section 4). The computer implementation is based on revolve developed by Griewank & Walther in [5]. It extends this package to adaptive checkpointing and is called a-revolve.

As far as the authors know there are only few contributions to control of the instationary Navier-Stokes equations that tackle the storage problem. The optimal static approach of revolve is applied to the Navier-Stokes system by Walther and the first author in [10], where also further references to the literature can be found. Optimal checkpointing for the Burgers equation is discussed in [17]. A general analytical framework for first and second order derivatives for control of the instationary Navier-Stokes equations is presented by Kunisch and the first author in [9].

This paper is organized as follows. Section 2 introduces checkpointing techniques and adaptive reversal schedules. In Section 3 the instationary Navier-Stokes equations are adapted to the setting introduced at the beginning of Section 1. Among other things the discretization concept for the forward and adjoint PDEs is introduced. Section 4 describes the implementation of adaptive checkpointing strategies and illustrates their capabilities w.r.t. memory reduction in the calculation of the adjoint states. Moreover, memory requirements and run-time results for both, the basic approach and the adaptive checkpointing approach for solving the adjoint PDE are shown. Furthermore, the adaptive approach is compared to the optimal static approach for non-uniform step cost w.r.t run time. Finally, in Section 5 we present some conclusions.

2 Formalism

In the present section we introduce a formal concept for optimal and adaptive checkpointing in the context of numerical integration of backward-in-time problems. In this context we refer to (2) and (4) as (the realization of a) (forward) PDE and adjoint PDE, respectively.

2.1 Subfunctions, taping and adjoint variables

As is shown in Section 1, adjoint operators are one basic ingredient of the derivatives $\hat{J}'(u)$ and $\hat{J}''(u)$. Their numerically realizations have to be based on appropriate discretizations of adjoint PDEs that have to be integrated backwards in time where the state variables

$y(u)$ enter as coefficients and right-hand-sides.

We now adapt the discretization process to the needs of the software tool a-revolve. Since a-revolve is an extension of revolve developed in [5], for the convenience of the reader we begin with presenting a brief introduction to checkpointing and reversal schedules follow the exposition of [10]. To begin with we note that for calculating an approximation of $y(u)$ corresponding to the control u one usually has to evaluate subfunctions F_j, $0 \leq j < l$, called time steps, that act on the state y^j to calculate the subsequent intermediate state y^{j+1} for $0 \leq j < l$ depending on a control action $\tilde{u}^j$, i.e.,

$$y^{j+1} \quad = \quad F_j(y^j, \tilde{u}^j)\ . \tag{6}$$

For the moment we assume that l is fixed and known. In order to compute a solution of the adjoint PDE the discretization yields adjoint time steps $\bar{F}_j$ for $l > j \geq 0$ with

$$\lambda^j \quad = \quad \lambda^{j+1}\, F_j'(y^j, \tilde{u}^j) \ \equiv \ \bar{F}_j(y^j, \tilde{u}^j, \lambda^{j+1})\ . \tag{7}$$

The evaluation of $\bar{F}_j$ may require some intermediate results calculated during the computation of y^{j+1} from the previous state y^j. Hence, it is supposed that for each $j \in \{0, \ldots, l-1\}$, there exist a recording step $\hat{F}_j$ that causes the recording of intermediate values required during the evaluation of the time step F_j onto a data structure called *tape*.

Using the recording step $\hat{F}_j$ and adjoint time step $\bar{F}_j$ the basic way to integrate the adjoint PDE reads as follows. First one evaluates $\hat{F}_j$ for $j = 0, \ldots, l-1$ in order to provide the intermediate values required for calculating an approximation of the solution of the adjoint PDE. Based on this data one evaluates $\bar{F}_j$ for $j = l-1, \ldots, 0$ to compute the desired adjoint information. This basic approach is illustrated by Fig. 1. As can be seen

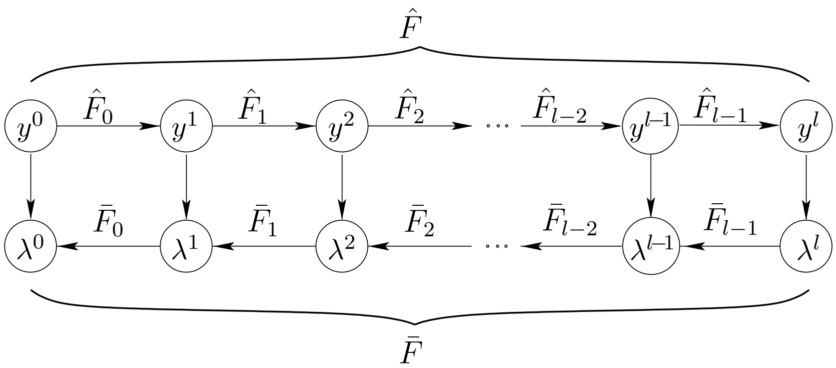

Figure 1: Basic approach to calculate Adjoints

the storage needed for the intermediate results is proportional to the number l of time steps. Therefore, applied to real-world problems with thousands of time steps the memory requirement of the basic approach becomes a problem even if only the intermediate states y^j, $j = 0, \ldots, l$ have to be saved on the tape. For example, computing 3D flows with unstructured grids one may need easily 10 to 100 MBytes to store one state vector y^j [12]. For that reason, it will also be assumed that due to their size, only a limited number of intermediate states can be kept in memory. Applying a checkpointing technique, the intermediate values are generated piecewise by restarting the evaluation repeatedly from suitably placed checkpoints, according to requests by the reversal process. Therefore, the checkpoints can be thought of as pointers to nodes representing intermediate states y^j. Using a checkpointing strategy, the calculation of F can be reversed, even in such cases where the basic method to calculate adjoints fails due to excessive memory requirement (see e.g. [5, 6]).

2.2 Reversals of Function Evaluations

Computing the solution of an adjoint PDE can be interpreted as the reversal of a function evaluation because of the backward integration described above. In order to reverse a given evaluation procedure one may employ reversal schedules. These reversal schedules can be formalized as follows.

Definition 2.1 (Reversal Schedule S). Assume the function evaluation to be reversed comprises the time steps F_j, $j = 0, \ldots, l-1$. Let $c \in \mathbb{N}$ checkpoints be available each of which can accommodate one state vector. Then a *reversal schedule* S initializes $j = 0$ and subsequently performs a sequence of basic actions

$$\begin{array}{lcl} A_m & \equiv & \text{Increment } j \text{ by } m \in \{1, \ldots, l-j-1\} \\ D & \equiv & \text{Decrement } l \text{ by 1 if } j = l-1 \\ W_i & \equiv & \text{Copy state } j \text{ to checkpoint } i \in \{1, 2, \ldots, c\} \\ R_i & \equiv & \text{Reset state } j \text{ to checkpoint } i \end{array}$$

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

This definition of a reversal schedule relies closely on the one described in Chapter 12 of [4]. The action A_m corresponds to a forward sweep, i.e., the time steps $F_k, j \le k < j+m$, are performed. The execution of a recording step $\hat{F}_j$ and an adjoint time step $\bar{F}_j$ are combined to the basic action D in order to reverse the currently final time step. A checkpoint writing, i.e. the storage of one state vector, is done by W_i. The checkpoint reading, i.e. the recovering of one state vector, is performed by R_i.

Provided $c \ge 1$, it is always possible to find a finite reversal schedule for a given evolution $F_0, \ldots, F_{l-1}$. For example, one may use the reversal schedule S determined by the sequence

$$\begin{aligned} S \equiv\ & W_1 + A_{l-1} + D + R_1 + A_{l-2} + D + R_1 + \cdots + A_k + D + R_1 + \\ & \cdots + A_1 + D + R_1 + D\ , \end{aligned}$$

where + indicates successive performance and l denotes the given length.

The set of reversal schedules, which can be applied to reverse a sequence of l time steps using c checkpoints, has infinitely many members. Certainly, we are interested in those of them, which reverse the given step sequence using minimum of time. We called these reversal schedules as optimal reversal schedules. To derive an optimal reversal of the given evaluation procedure, one has to take into account four kinds of parameters, namely:

1.) the number l of time steps to be reversed;

2.) the number c of checkpoints that can be accommodated;

3.) the number p of processors that are available; and

4.) the step cost: $\tau_j = TIME(F_j)$, $\hat{\tau}_j = TIME(\hat{F}_j)$, $\bar{\tau}_j = TIME(\bar{F}_j)$.

Next let us introduce some notations, compare [4, 16]. Step cost are called **uniform** if $\tau = \tau_j$ for all $0 \leq j < l$, and **non-uniform** if $\tau_i \neq \tau_j$ for some $i \neq j$. Reversal schedules for a given number of time steps are called **static reversal schedules**. According to [14] we denote by $S_{bin}(l, c)$ and $S_{opt}(\mathbf{t}_l, c)$ the **optimal static reversal schedules** for uniform and non-uniform step cost, respectively, where $\mathbf{t}_l =< t_0, ..., t_{l-1} >$ denotes a step cost distribution in the non-uniform case. Optimal reversal schedules for one-processor machines (also called serial machines) are investigated in [4] for uniform step cost. For an extension of this work to non-uniform step costs see [16].

These reversal schedules allow an enormous reduction of the memory required to reverse a given evolution F in comparison with the basic approach (see e.g. [4, 16]). However, for both uniform and non-uniform step cost, one has to pay for the improvements in the form of a greater temporal complexity because of repeated forward integrations. For this reason, reversal schedules for multi-processor machines have been developed and implemented [13]. Nevertheless, in the present paper we consider only serial reversal schedules.

For the convenience of the reader the performance of $S_{bin}(10, 3)$ is presented in Fig. 2.

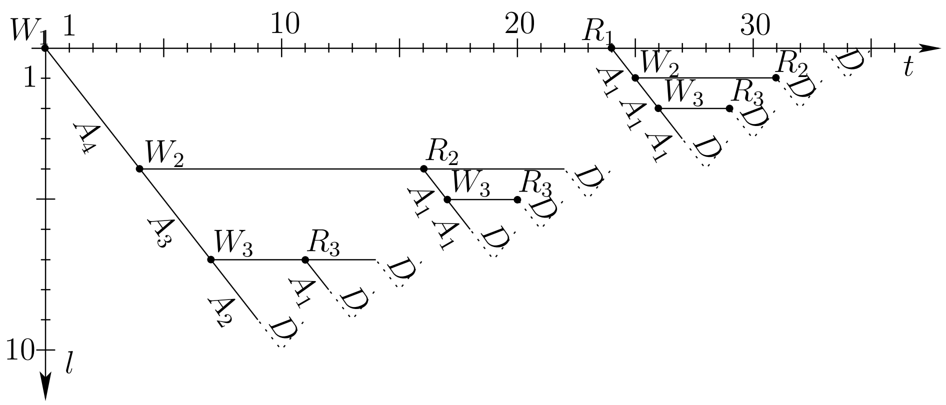

Figure 2: Optimal Distribution $S_{bin}(10, 3)$ of Actions for $l = 10$ and $c = 3$

Here, the time steps are plotted along the vertical axis and the computing time required to reverse F 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.

To anticipate discussion let us note that in our algorithmic implementation of adaptive reversal schedules it is required to determine certain sequences of static reversal schedules. Since in the case of non-uniform step cost the complexity of determining $S_{opt}(\mathbf{t_l}, c)$ is $\mathcal{O}(l^2c)$ the second author for this case in [14] developed a heuristic static reversal schedule which proves very efficient in many numerical tests. This static reversal schedule from now onwards is denoted by $S_{ef}(t_l, c)$ and we note that this schedule is the basic ingredient of the adaptive reversal schedules for non-uniform cases presented in the subsequent section. However, it could not be proved that the complexity of determining $S_{ef}(t_l, c)$ is $\mathcal{O}(lc)$, which is indicated by many numerical tests, see [14] for many examples.

Further, we will use the cost-efficient reversal schedules $S_{ef}(\mathbf{t_l}, c)$ in the construction of adaptive reversal schedules for non-uniform step cost.

2.3 Adaptive Reversal Schedules

We now extend the concept of revolve to situations where the number of time steps is not known in advance. In such cases the derivation of reversal schedules has to be performed adaptively. In this paper we investigate the adaptive reversal schedules for serial machines. Similar to the case of static reversal schedules one distinguishes **adaptive reversal schedules with uniform and non-uniform step cost**. Nevertheless, the general approach for the derivation of adaptive reversal schedules is similar in the both cases. We denote by l the a-priori unknown number of time steps, by c the number of available checkpoints, and by s the number of stored checkpoints. Obviously, $0 \leq s \leq c$ is. The following algorithm presents the basic algorithmical representation of adaptive reversal schedules for serial machines.

Algorithm 2.2. (Adaptive Reversal Schedules)

0. Set $l = 0$, $s = 0$; $c \in \mathbb{N}_+$ given.
1. **Initial Step**

   Store the data of the initial state as a first checkpoint.

   $s = s + 1$.
2. **Forward** & **Control Step**

   If F_l is not the last time step, perform the F_l-th time step, i.e. evaluate the intermediate state y_{l+1}, set $l = l + 1$.

   Else Go to 3.

   If $s < c$, set $s = s + 1$ and store the data of the state y_l in the checkpoint s.

   Go to 2.

<u>Else</u> check for optimality of the current checkpoint distribution. If it could be improved through substituting the content of a checkpoint by the data of the intermediate state y_l, replace the content of the corresponding checkpoint by y_l. Go to 2.

3. **Reversal Step**

   Initialize the reversal sweep and reverse the time steps $F_l, ..., F_0$.

$\square$

Let us briefly discuss the algorithm. In the initial step data of the first state y_0 is stored in the first checkpoint. Obviously, this step of the algorithm is identical for both, uniform and non-uniform step cost.

In the forward and control step of the algorithm we assume that at least $(l + 1)$ time steps F_0,...,F_l have to be reversed. It is clear, that before we start to check for the optimality of the checkpoint distribution and eventually replace contents of checkpoints, we store the data of the first states $y_0, ..., y_{c-1}$ in the checkpoints $1, .., c$. So far, if a step sequence has at least c time steps $F_0, ..., F_{c-1}$, all given checkpoints are occupied after performing of time steps $F_0, ..., F_{c-2}$. These actions corresponded to the second **<u>If</u>** of the **Forward & Control Step** in the Algorithm 2.2 and of cause are identical for the both, uniform and non-uniform step cost.

The set $\{d_1, ..., d_s\}$ denotes the checkpoint distribution, where d_i, $1 \leq i \leq s$, is the number of the intermediate state, whose data is stored in the i-th checkpoint. The data of the currently evaluated intermediate state y_{l+1} is available as well.

Now the content of some checkpoint i, $2 \leq i \leq s$, can be overwritten with the data of the intermediate state y_{l+1} that is currently available. Then the sequence of time steps performed between the two last checkpoints can be reversed using the now available two checkpoints, so that the overall cost for the reversal of all time steps F_0,...,F_l can be reduced.

Such a reallocation of the checkpoint distribution has to be performed if there is an index i, $2 \leq i \leq s$, such that

$$TIME(d_1, ..., d_{i-1}, l, d_{i+1}, ...d_s) < TIME(d_1, ..., d_{i-1}, d_i, d_{i+1}, ...d_s), \tag{8}$$

where the right hand side of (8) denotes the cost for the reversal of time steps F_0,...,F_l using the current checkpoint distribution $\{d_1, ..., d_s\}$. The left hand side of (8) is associated to the reversal cost if the i-th checkpoint is reallocated with y_l. We assume that the subsequences of time steps performed between the appropriate checkpoints are reversed using optimal reversal schedules $S_{bin}(d_{j+1} - d_j, c - j + 1)$, $j = 1, ..., s - 1$ and $S_{bin}(l - d_s, 1)$ in the case of uniform step cost, and using reversal schedules $S_{ef}(\mathbf{t_{d_j+1,d_{j+1}}}, c - j + 1)$, $j = 1, ..., s - 1$ and $S_{ef}(\mathbf{t_{d_s+1,l}}, 1)$ for non-uniform step cost (for details see [14]). The control of the checkpoint distribution described in (8) is implemented accordingly. The first and the second step of the Algorithm 2.2 are illustrated for an example with $c = 4$ checkpoints in Fig. 3.

In order to reverse the step sequence $F_0, ..., F_l$, $l \geq 5$, we first store data of y_0 in the $1-$st checkpoint according to the **Initial Step** of Alg. 2.2. Then $d_1 = 0$. After that we perform steps F_0, F_1 and F_2 and store the data of the evaluated states y_1, y_2 and y_3 in the remaining checkpoints, so that $d_2 = 1$, $d_3 = 2$ and $d_4 = 3$. This situation is sketched in

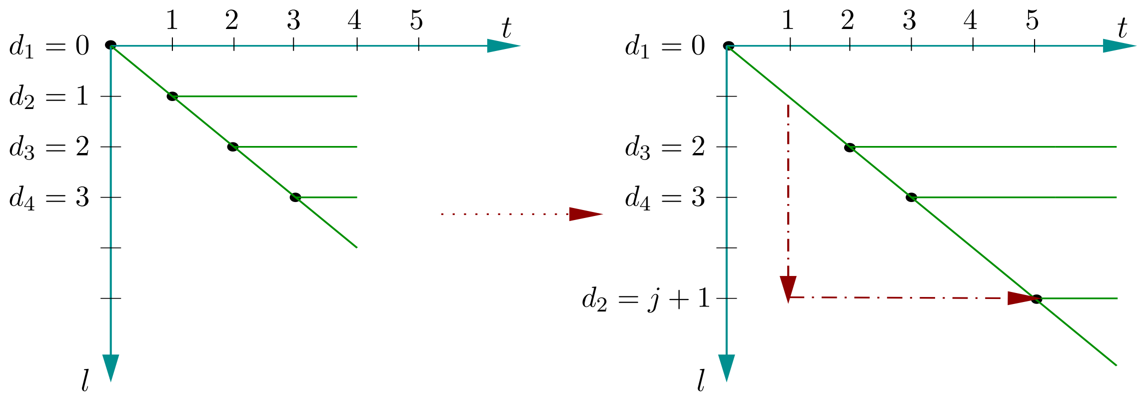

Figure 3: An example of checkpoint reallocation

Fig. 3, left. After every evaluation of F_j, $j = 3, \ldots$ we check for optimality of the current checkpoint distribution $\{0, 1, 2, 3\}$ using (8). If for some j the substitution of the content of the checkpoint y_k $(0 \le k \le 3)$ through the data of y_j would lead to a reduction of the overall evaluation cost y_k is replaced by y_j. For details dealing with the selection rules for y_k we refer to [14]. The situation where it is most convenient to overwrite the contents of the second checkpoint with the data of y_j for some $j \ge 3$ is illustrated in Fig. 3, right. Now repeat this procedure in the **Forward & Control Step** till the last time step is performed and the algorithm enters the **Reversal Step** of Alg. 2.2.

When entering the **Reversal Step** we have available a step sequence $F_0, \ldots, F_l$ (where l is now a known value), a checkpoint distribution $\{d_1, \ldots, d_s\}$, and the state y_l, and thus we are in the situation for applying a static reversal schedule as described in Section 2.2. Note that $s < c$ only if $l < c$. First we perform the recording and the adjoint step $\hat{F}_l$ and $\bar{F}_l$, respectively. Then we reverse the step sequences $\{F_{d_s}, \ldots, F_{l-1}\}$, $\{F_{d_{s-1}}, \ldots, F_{d_s-1}\}$, $\ldots$, $\{F_{d_1}, \ldots, F_{d_2-1}\}$, using S_{bin} for uniform and S_{ef} for non-uniform step cost, respectively.

We note that each adaptive reversal schedule constructed according to Alg. 2.2 can be represented as a sequence of the basic actions A_m, D, W_i and R_i defined Def. 2.1.

From now onwards we denote by $S_{adapt}(l, c)$ and $S_{adapt}(\mathbf{t_l}, c)$ the adaptive reversal schedules for uniform and non-uniform step cost, respectively.

An example of an adaptive reversal schedule $S_{adapt}(10, 3)$ for $l = 10$ and $c = 3$ with uniform step cost is shown in Fig. 4.

Here, as in Fig. 2 the time steps are plotted along the vertical axis and the computing time required to reverse F is represented by the horizontal axis. As can be seen, only one checkpoint reallocation is performed in this example. The content of the second checkpoint is substituted once, i. e. its data is replaced by that of the intermediate state y_6 after performing the F_5-th time step. After the evaluation of y_9, it is determined, that the step sequence contains 10 time steps $F_0, \ldots, F_9$. Then the steps $\hat{F}_9$ and $\bar{F}_9$ are performed and the subsequences $\{F_6, F_7, F_8\}$, $\{F_2, F_3, F_4, F_5\}$ and $\{F_0, F_1\}$ are reversed using the optimal reversal schedules $S_{bin}(3, 1)$, $S_{bin}(4, 2)$ and $S_{bin}(2, 3)$, respectively.

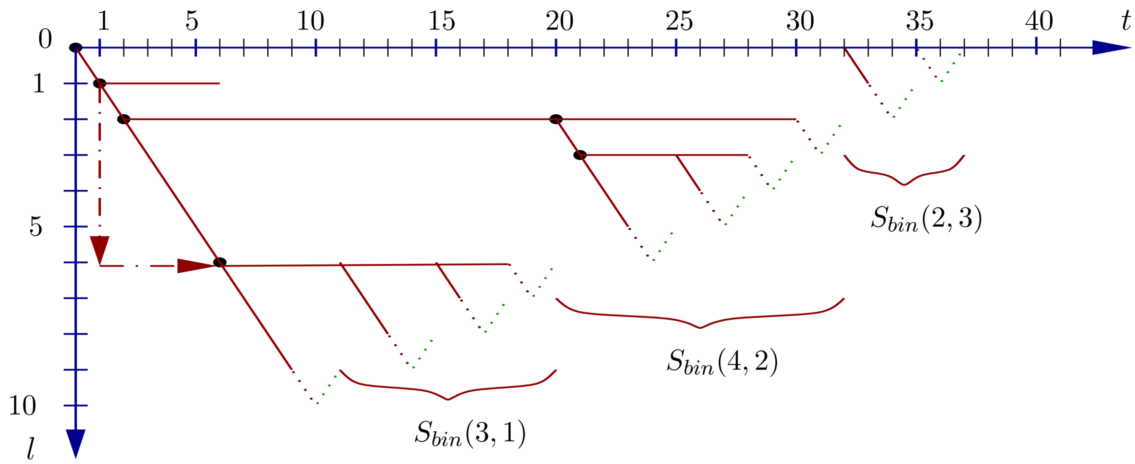

Figure 4: An example of adaptive reversal schedule $S_{adapt}(10, 3)$

3 Application to the instationary 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 algorithmically one we only introduce the solenoidal spaces

$$H := \{v \in L^2(\Omega)^{2,3}, \text{div } v = 0\}^{clos_{L^2}} \text{ and } V := \{v \in L^2(\Omega)^{2,3}, \text{div } v = 0\}^{clos_{H^1}}$$

and omit the introduction of a more detailed functional analytic setting. Interested readers are referred to [8, 9], and also to [10].

From now 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

$$\left.\begin{array}{rcl} y_t - \nu\Delta y + (y \cdot \nabla)y + \nabla p &=& Bu \text{ in } Q, \\ -\text{div } y &=& 0 \text{ in } Q, \\ y(x,t) &=& 0 \text{ on } \partial\Omega \times (0,T), \\ y(x,0) &=& y_0(x) \text{ in } \Omega \end{array}\right\} \tag{9}$$

is satisfied, where $y_0 \in H$ denotes the initial value. Here $Q := (0,T) \times \Omega$ 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 [15] of Temam.

In order to express the actions of $\hat{J}'(u)$ and $\hat{J}''(u)$ for the cost functional given in eq. (1) 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 $\lambda \in Z$ formally be defined by

$$\lambda \;=\; (\lambda^1, \lambda^2) \;=\; -G_y(y,u)^{-*} J_y(y,u), \tag{10}$$

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

$$v := G_y(y,u)^{-1}(f, v_0). \tag{11}$$

It is shown in [9, 10] that these formal equations in the primitive setting allow the following interpretations; together with the adjoint pressure ξ the adjoint variables λ of (10) satisfy the system

$$\left.\begin{array}{rcl} -\lambda^1_t - \nu\Delta\lambda^1 - (y\cdot\nabla)\lambda^1 + (\nabla y)^t\lambda^1 + \nabla\xi &=& -J^{(t)}_{1_y}(y) \text{ in } Q, \\ -\text{div } \lambda^1 &=& 0 \text{ in } Q, \\ \lambda^1(x,t) &=& 0 \text{ on } \partial\Omega\times(0,T), \\ \lambda^1(x,T) &=& -J^{(T)}_{1_y}(y) \text{ in } \Omega, \end{array}\right\} \tag{12}$$

and $\lambda^2 = \lambda^1(0)$. The superscripts $(t), (T)$ refer to a possible dependence of the representative of the functional J_{1_y} on the time instances (t) and (T), respectively. Finally, together with some pressure ρ the function v of (11) satisfies

$$\left.\begin{array}{rcl} v_t - \nu\Delta v + (y\cdot\nabla)v + (v\cdot\nabla)y + \nabla\rho &=& f \text{ in } Q, \\ -\text{div } v &=& 0 \text{ in } Q, \\ v(x,t) &=& 0 \text{ on } \partial\Omega\times(0,T), \\ v(x,0) &=& v_0(x) \text{ in } \Omega. \end{array}\right\} \tag{13}$$

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 (9) 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

$$\Delta t_j = 0.7\frac{h}{\max_{x\in\Omega}|y(t_j)|}, \tag{14}$$

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 by the way are the minimal regularity requirements for proving error estimates for numerical approximation schemes of the Navier-Stokes system, compare [3, 7]. 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 ($y^j \sim y(t_j)$) y^j with $0 \le j < l$ is given by

$$
\begin{aligned}
\tfrac{y^{j+1}-y^j}{\Delta t_j} - \nu\Delta y^{j+1} + \nabla p^{j+1} &= (Bu)^j - (y^j \nabla y^j) \text{ in } \Omega \\
-\operatorname{div} y^{j+1} &= 0 \text{ in } \Omega,\ y^{j+1} = 0 \text{ in } \Omega
\end{aligned}
\tag{15}
$$

with $y^0 = y(0)$. In the setting of Section 2.2 this forward integration scheme may be rewritten as

$$
y^{j+1} = F_j(y^j, \tilde{u}^j), \tag{16}
$$

where the time step function F_j is given by $F_j(y^j, \tilde{u}^j) = F(\Delta t_j, y^j, (Bu)^j)$ with

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

Here, S denotes the Stokes Operator and P defines the Leray projection $L^2(\Omega)^{2,3} \to H$, see [2]. As can be seen, equation (16) matches exactly the time stepping (6) that form the basis for the reversal schedules presented in Section 2.

The time discretization scheme of the adjoint variables λ in (12) is more involved. Here we take the transpose of the semi-implicit time discretization of eqs. (13) given next; for $0 \le j < l$ let

$$
\begin{aligned}
\tfrac{v^{j+1}}{\Delta t_j} - \nu\Delta v^{j+1} + \nabla \rho^{j+1} &= f^j + \tfrac{v^j}{\Delta t_j} - (y^j \nabla v^j) - (v^j \nabla y^j) \text{ in } \Omega \\
-\operatorname{div} v^{j+1} &= 0 \text{ in } \Omega,\ v^{j+1} = 0 \text{ on } \partial\Omega,
\end{aligned}
\tag{17}
$$

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

$$
\begin{aligned}
\tfrac{v^0}{\Delta t_0} - \nu\Delta v^0 + \nabla \rho^0 &= \tfrac{v_0}{\Delta t_0} - \nu\Delta v_0 \text{ in } \Omega \\
-\operatorname{div} v^0 &= 0 \text{ in } \Omega,\ v^0 = 0 \text{ on } \partial\Omega,
\end{aligned}
\tag{18}
$$

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

Formally transposing the scheme (17), (18) we obtain for the adjoint equations and $j = l-k \ge 0$ the integration scheme

$$
\begin{aligned}
\tfrac{\lambda^{l-k}}{\Delta t_{l-k}} - \nu\Delta\lambda^{l-k} &= \tfrac{\lambda^{l-k+1}}{\Delta t_{l-k}} - \nabla \xi^{l-k} - J_{1_y}^{(t_{l-k})}(y^{l-k}) \\
&\quad + (\lambda^{l-k+1} \nabla y^{l-k}) - (\nabla y^{l-k})^t \lambda^{l-k+1} \text{ in } \Omega, \\
-\operatorname{div} \lambda^{l-k} &= 0 \text{ in } \Omega,\ \lambda^{l-k} = 0 \text{ on } \partial\Omega,
\end{aligned}
\tag{19}
$$

where the states y^{l-k} are given by the solution of (15) and $\lambda^i \equiv 0$ for all $i \ge l+1$. We note, that the initial state λ^l in this integration scheme satisfies the quasi-Stokes system

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

This adjoint integration scheme may be rewritten as

$$\lambda^j = \bar{F}(\Delta t_j, y^j, \lambda^{j+1}), \tag{20}$$

where

$$\bar{F}(\Delta t, y, z) := \Delta t \, (P - \nu \Delta t S)^{-1} \left(-J^{(t)}_{1_y}(y) + (z\nabla)\, y - (\nabla y)^t \, z \right) + Pz. \tag{21}$$

If we set

$$\bar{F}_j(y^j, \tilde{w}^j, \lambda^{j+1}) := \bar{F}(\Delta t_j, y^j, \lambda^{j+1}),$$

(20) exactly matches (7), e.g. the adjoint time steps fit exactly for the application of the reversal schedules. Furthermore, one finds that the recording step $\hat{F}(y^j, (Bu)^j)$ consists of the evaluation of $F(\Delta t_j, y^j, (Bu)^j)$ and the storage of the new state vector y^{j+1}.

Next let us briefly discuss the time discretization of $\hat{J}''(u)\delta u$ described in Algorithm 1.1. 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 Algorithm 1.1 we propose to apply scheme (17) for the computation of v with $v_0 \equiv 0$, with y taken from (15). We recall that the adaptive time grid obtained from (14) is also used in this scheme. In step 2. the numerical results of schemes (15) for y and (19) for λ are utilized to evaluate rhs. Finally, for the numerical computation of μ in step 3. we propose to use again scheme (19), 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.

Remark 3.1. The application of the time-discretization scheme (19) for the adjoint equations among other things guarantees symmetry of a so discretized reduced Hessian, compare (5). It also immediately follows from this approach that the states y in the convective terms in (19) are taken at the same time instances as the unknown adjoint variables.

Of course is the discretization principle for the adjoint equations and reduced Hessian times increment presented above not restricted to semi-implicit time integration schemes. Its extension to more sophisticated time integration schemes is straightforward.

Remark 3.2. Since in general the time step size Δt_j is different for every time step and $\tau_j \sim \Delta t_j$, i.e. the cost of step F_j is proportional to the step size Δt_j. This can be explained as follows. If one solves the system (17) by a preconditioned Schur-complement CG-method, say as it is the case in the numerical part of the present work the performance of the method heavily depends on the size of $\frac{1}{\Delta t_j \nu}$, see [11]. Our numerical test further indicate, that the numerical effort for the integration step is proportional to the size of Δt_j. Therefore, adaptive reversal schedules for problems with non-uniform step cost have to be applied to reverse (16). We note that for the calculation of the adjoint the time grid determined through the forward & control step of Algorithm 2.2 is utilized. If one would apply adaptive time stepping also to the calculations of the adjoints appropriate interpolation routines for the state variables need to be incorporated into the reversal step of the algorithm.

We note that the steps of the discretization procedure above have also to be performed for the static approach of revolve and also for the basic approach (storing all states during a forward & control run). The following section presents how the adaptive reversal schedules described in Section 2 can be applied to compute adjoints for the instationary Navier-Stokes system. Specifically, emphasize is taken on the enormous reduction of memory requirement that is achieved by the approach presented.

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 1, i.e., $T = 1$, and $\nu = 1/\mathrm{Re}$ with Re$= 10$. The vector

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

with e denoting the Euler number is used as initial condition. Now, the goal is to approximate the time-dependent desired state given by

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

Here, φ is defined through the stream function

$$\varphi(x_1,x_2,t) = \theta(x_1,t)\theta(x_2,t)$$

with

$$\theta(y,t) = (1-y)^2(1-\cos 2k\pi t), \quad y \in [0,1].$$

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

$$\hat{J}(u) = J(y,u) \quad = \quad \frac{1}{2}\int_0^T \int_\Omega |y-z|^2 dx\, dt + \frac{c}{2}\int_0^T \int_\Omega |u|^2 dx\, dt \tag{22}$$

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

As mentioned above, using the recording steps $\hat{F}_j(y^j,(Bu)^j)$ of (16) and the adjoint steps $\bar{F}_j(y^j,\tilde{u}^j,\lambda^{j+1})$ of (21) one may compute the gradient $\hat{J}'(u)$ applying the basic approach as illustrated in Fig. 1. The basic algorithm then reads as follows:

init(y), $t_0 = 0$, Δt_0 from (14) with $j = 0$, $t = \Delta t_0$.
While $t < T$
  hF(y,u)
  $j = j + 1$
  Δt_j from (14)

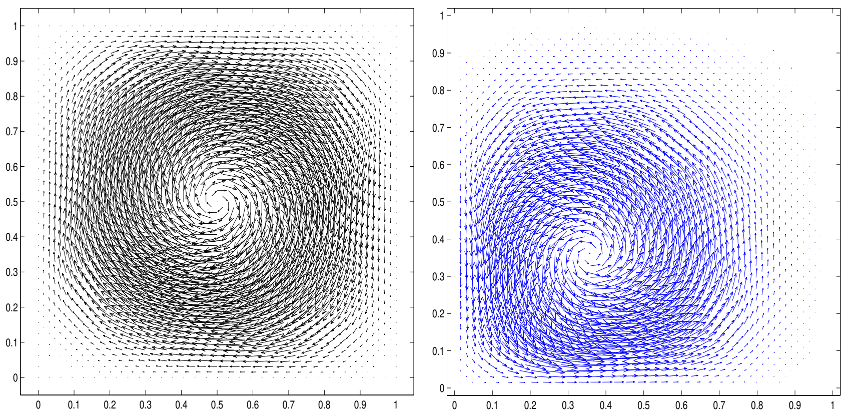

Figure 5: Left: Cavity flow at $t = 0.01$, right: Desired flow at $T = 1$

$t = t+$ min$(\Delta t_j, T - t)$
$l = j + 1$
init(lam)
for $j = l - 1, \ldots, 0$
bF(y,u,lam)

where hF(y,u) corresponds to $\hat{F}_j(y^j, (Bu)^j)$ and bF(y,u,lam) to $\bar{F}_j(y^j, \tilde{u}^j, \lambda^{j+1})$. Using the basic approach, a complete trajectory is stored during the forward integration in the while-loop. This trajectory is read backward during the adjoint calculation in the for-loop. Using a discretization with 2113 velocity nodes and 545 pressure nodes, the storage of the full state information causes a memory requirement of 3.8 MByte for $l = 100$ time steps. If 10.000 time steps have to be performed the memory requirement would be equal to 380 MByte. Note, that only a 2D problem is considered here. For computing the solution of a 3D problem the situation gets even worse. The adaptive reversal schedules presented in Section 2 offer an opportunity to reduce this memory requirement drastically. For this purpose, the routine a-revolve may be applied. The corresponding algorithm reads as follows:

snaps = c capo = 0 fine = 0 check = -1 turn = 0 $t = \Delta t_0$
Δt_0 from (14) with $j = 0$
do
$\tau_{capo} = Cost(\Delta t_{capo})$, Δt_{capo} from (14) with $j = capo$
whatodo = a-revolve(check,capo,fine,τ_{capo},w)
switch(whatodo)

```
        case advance:   F(y,u)
                        if (turn = 0)
                           Δt_capo from (14) with j = capo
                           t = t+ min(Δt_capo, T − t)
        case takeshot:  store(y,ystor,check)
        case firsturn:  hF(y,u)
                        init(lam)
                        bF(y,u,lam)
        case youturn:   hF(y,u)
                        bF(y,u,lam)
        case restore:   restore(y,ystor,check)
        case error:     print(schedule error!)
    end switch
    if (turn = 0) and (t < T)
       fine = fine+1
    else turn = 1
 while((whatodo <> terminate) && (whatodo <> error))
```

The first line contains initializations. Subsequently, the 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 variable snaps denotes the number of available checkpoints and check the number of actually used checkpoints. τ_{capo} denotes the cost of the time step F_{capo}. In our example τ_{capo} is set equal to the time step size, $\tau_{capo} = \Delta t_{capo}$. This corresponds to $Cost(\Delta t) = \Delta t$. By w an array of cost is denoted, which stores the cost 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 the Definition 2.1 of reversal schedules. The return value advance corresponds to the basic action A_1, and the return value takeshot to the basic action W_{check}. If some data was already stored in the checkpoint check, then this data has to be replaced by the data currently available from the intermediate state y_{capo}. The return value restore is equal to R_{check}. For the decrement action D, a-revolve distinguishes between the first decrement and further decrements. In the first case, a-revolve returns firsturn so that the initializations of the adjoint variables may be performed. In the latter case, a-revolve returns youturn.

As can be seen, the do-while-loop is completely independent of the actual problem to be reversed. The functions F(y,u), hF(y,u), and bF(y,u) are required also for the basic approach to calculate 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 adjoint calculation.

In the remainder of this section we investigate the numerical performance of a-revolve. Fig. 6 shows an example of checkpoint distributions and reallocations for the instationary Navier-Stokes equations while the function $F = F_0 + ... + F_{26}$ was evaluated. For spatial

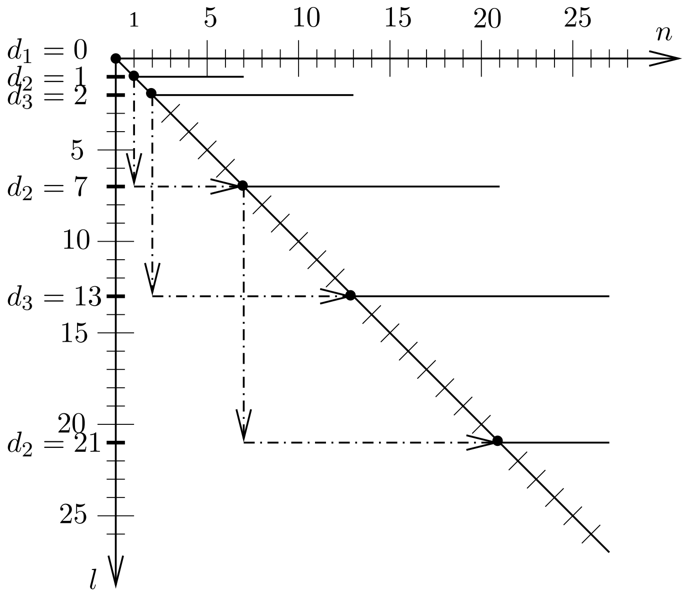

Figure 6: Example of checkpoint reallocations for the instationary Navier-Stokes Equations (l = 27, c = 3, spatial discretisation: 41 v-nodes, 13 p-nodes, 16 triangles)

discretisation a grid with 16 triangles on the time horizon $[0, T] = [0, 1]$ is used. This corresponds to 41 velocity and 13 pressure nodes. For storage 3 checkpoints are available, i.e. $c = 3$. The time steps are plotted along the vertical axis. Solid horizontal lines including the x-axis itself represent checkpoints. Since the step cost τ_i, $0 \leq i < 27$ in general differ for different values of i, the number n of currently evaluated time steps is plotted along the horizontal axis, so that this axis no longer can be thought of as the computational axis as it is the case in Fig. 2. Checkpoint reallocations are depicted by dashed-dotted lines.

For this test run Alg. 2.2 works as follows. At the beginning the data of the initial intermediate state y_0 and of the computed intermediate states y_1, y_2 are stored in the checkpoints. The efficiency of the actual checkpoint distribution is checked after every performed time step using (8). During the forward sweep three checkpoint reallocations are performed in total: the content of the second checkpoint is replaced twice, and the third checkpoint once. The total number of time steps in this example is 27. Finally, after performing of the 25−th time step the data of the y_0-th, y_{13}-th and y_{21}-th intermediate states are stored in the three available checkpoints.

Fig. 7 shows an evaluation cost ratio for adaptive reversal schedules applied for computation of the gradient $\hat{J}'(u)$ of the functional in (22). The evaluation cost for adaptive reversal schedules for several values of c is compared to the evaluation cost of static optimal reversal schedules with non-uniform step cost using a step cost distribution obtained from one forward computation with time-stepping according to (14). This ratio is evaluated

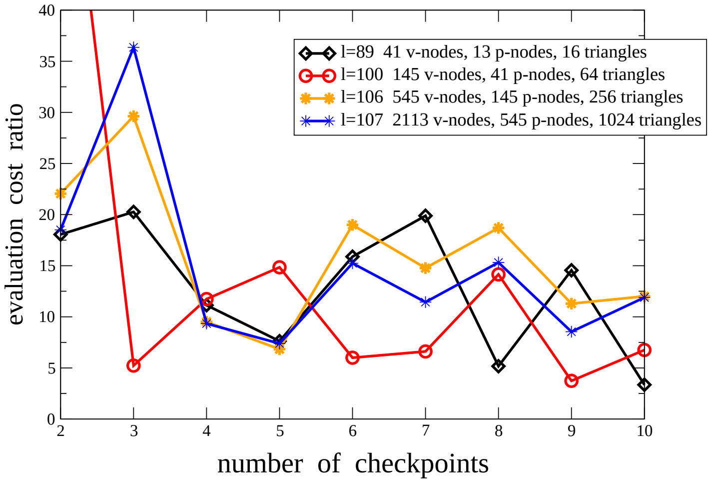

Figure 7: Evaluation cost needed to compute $\hat{J}'(u)$ using adaptive reversal schedules compared to that needed when using static optimal reversal schedules for non-uniform step cost

.

using the formula

$$\frac{T(S_{adapt}(\mathbf{t}_1, c)) - T(S_{opt}(\mathbf{t}_1, c))}{T(S_{opt}(\mathbf{t}_1, c))} * 100, \tag{23}$$

where $T(S_{adapt}(\mathbf{t}_1, c))$ and $T(S_{opt}(\mathbf{t}_1, c))$ denote the cost needed to evaluate adaptive and static optimal reversal schedules for non-uniform step-cost distribution $\mathbf{t}_1$, respectively and c checkpoints. As **evaluation cost** we denote the sum of the costs τ_i over all time steps F_i which need to be evaluated for reversal of the corresponding step sequence. The vertical axis shows the ratio of the evaluation cost. As can be seen $T(S_{adapt}(\mathbf{t}_1, c)) - T(S_{opt}(\mathbf{t}_1, c))$ varies between 5 and 20 % relative to the optimal evaluation cost $T(S_{opt}(\mathbf{t}_1, c))$ for $4 \leq c \leq 10$ checkpoints. In order to present numerical tests for adaptive reversal schedules with a reasonable number of time steps, the time horizon in the following examples is set to $[0, T] = [0, 40]$.

The desired gradient $\hat{J}'(u)$ is computed applying the adaptive reversal algorithm described above for varying numbers of checkpoints. Fig. 8 shows the observed run-time behaviour for different spatial discretizations. Here, the vertical axis shows the ratio of the run-time needed to compute $\hat{J}'(u)$ and that needed to compute $\hat{J}(u)$. Using the basic approach for computing adjoints, the upper bound for this ratio varies between three and five depending on how to count operations and memory accesses [4]. The horizontal axis denotes the number of checkpoints used by the reversal schedule.

As can be seen, the run-time ratio varies between 3 and 5 for a reasonable number of

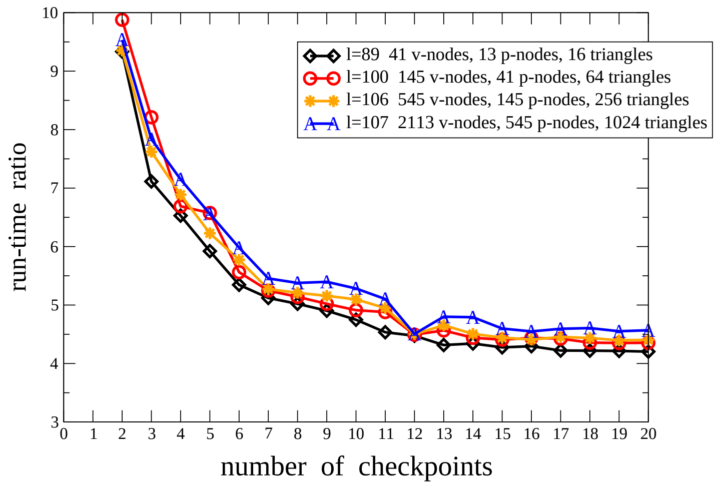

Figure 8: Run-time needed to compute $\hat{J}'(u)$ using adaptive reversal schedules compared to that needed to compute an evaluation of the functional $\hat{J}(u)$

.

checkpoints. This behaviour results in a slow down factor up to 2 compared to the basic approach, where a complete trajectory is stored to compute the adjoint values. Nevertheless, the memory requirement is reduced enormously. Using a spatial discretization with 2113 velocity and 545 pressure nodes one needs 38 kByte to store one checkpoint. Hence, if the reversal schedule utilizes 6 checkpoints, the memory requirement equals 228 kByte for calculating adjoints with reversal schedules. If the basic approach is applied, the memory requirement amounts to 3.8 MByte, compare also [10].

From Fig. 8 we conclude that the dependence of the run time ratio on the spatial discretization indeed only is mild, if there is any dependence at all. We further note that the results obtained with revolve in [10] are very similar. For a given memory requirement (=number of checkpoints) the performance with respect to the required run-time is good.

Since the resulting variations lie around a certain run-time ratio if the number of checkpoints exceeds an lower bound one may draw the conclusion that the number of checkpoints is not the main influence on the run-time behaviour. Hence above the lower bound, the number of checkpoints can be varied without having a big influence on the run-time. The fact is also illustrated by the flat development of the run-time ratios in Fig. 8 if the number of checkpoints exceeds twelve.

Since the number of time steps to reverse is always unknown in advance for the problems investigated here, it is useful to compare the performance of the adaptive schedule to that of the optimal static approach with a time step sequence generated by the forward integration of the PDE. Fig. 9 shows the observed run-time ratio behavior for the already investigated problems. The vertical axis there gives the ratio of the run-time needed to compute $\hat{J}'(u)$

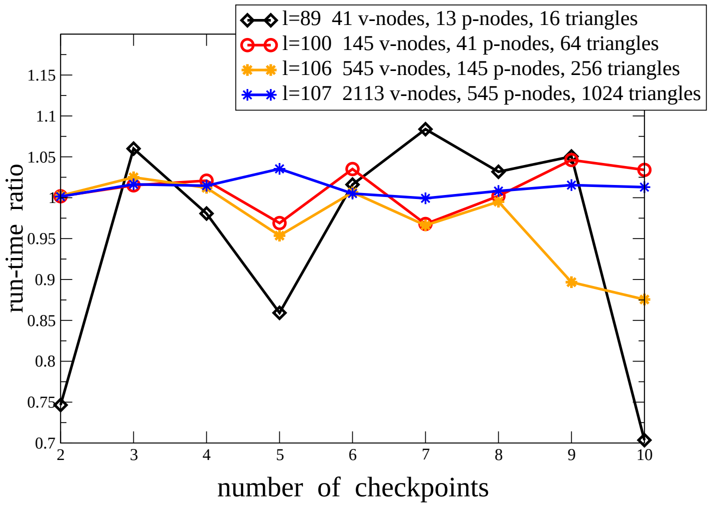

Figure 9: Run-time needed to compute $\hat{J}'(u)$ using adaptive reversal schedules compared to that needed for the static optimal reversal schedules with non-uniform step cost

using adaptive and optimal reversal schedules. This ratio is evaluated using the formula

$$\frac{CLOCK(S_{adapt}(\mathbf{t_l}, c))}{CLOCK(S_{opt}(\mathbf{t_l}, c))}, \tag{24}$$

where $CLOCK(S_{adapt}(\mathbf{t_l}, c))$ and $CLOCK(S_{opt}(\mathbf{t_l}, c))$ are run-times needed to evaluate adaptive and static optimal reversal schedules for non-uniform step-cost $\mathbf{t_l}$, respectively and c checkpoints. The horizontal axis denotes the number of checkpoints used by the reversal schedule. Zig-zagging here and the fact that the run-time lies sometimes under the 1 (supposing that the adaptive reversal schedules are "better" as the static optimal one) is due to other processes which are performed on our single processor machine during the run time of our function and gradient evaluations. Therefore, the run-time depends also naturally on the current load of the machine.

If we compare the theoretical run-times ratios from the Fig. 7 with the actually measured run-times ratios from the Fig. 9, we see that practically run-times needed for the computation of $\hat{J}'(u)$ using adaptive reversal schedules increase only slightly or even sometimes don't increase altogether compared to the run-times needed for the computation of $\hat{J}'(u)$ using static optimal reversal schedules with non-uniform step cost. It can be explained by the fact, that the most run-time by the computation of $\hat{J}'(u)$ will be required for the loading and access of memory.

As can be seen, the run-time ratio is very close to that of the optimal reversal schedules. So far, for the numerical examples presented here the adaptive reversal schedules cause only a slight increase of the run-time cost compared with the static optimal checkpointing for non-uniform step cost.

5 Conclusions

For 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 an adaptive reversal schedule that allows a drastic reduction of the memory requirement while run-time compared with the basic approach increases only slightly. Moreover, adaptive checkpointing causes only a slight increase of the run-time compared with the static optimal checkpointing for uniform and non-uniform step cost. This is illustrated by comparing the performance of both approaches at numerically computing adjoints of the instationary Navier-Stokes system. Moreover, as is illustrated in [14] the adaptive reversal schedules presented in this paper clearly outperforms the naive approach, which firstly would determine the number l of time steps to revers by applying a forward sweep without storing intermediate data, and then using the corresponding optimal reversal schedule for uniform or non-uniform step cost to reverse the process. The routine a-revolve, which is the extension of revolve mentioned in [10], implements such adaptive reversal schedules. The resulting memory reduction evaluation and run-time behaviour is studied for an optimal control problem based on incompressible Navier-Stokes equations. The achieved results are quite promising.

Further research should be devoted to the study of the dependence of the run-time ratios on the size of the discretization and the dependence of the step cost τ_j on the time discretization Δt_j.

Acknowledgments

Both authors acknowledge support of the Sonderforschungsbereich 609 **Elektromagnetische Strömungsbeeinflussung in Metallurgie, Kristallzüchtung und Elektorchemie**, funded by the Deutsche Forschungsgemeinschaft. The authors are grateful to Andreas Griewank and Andrea Walther from the Institute of Scientific Computing at the Technische Universität Dresden for many discussions on the topic of this work. The authors are also grateful to the referees for many valuable suggestions and comments.

References

[1] **Abergel, F.** & **Temam, R.** On some Control Problems in Fluid Mechanics. *Theoret. Comput. Fluid Dynamics*, 1:303–325, 1990.

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

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

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

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

[6] **Grimm, J., Pottier, L.,** & **Rostaing-Schmidt, N.** 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.

[7] **Heywood, J.G.** & **Rannacher, R.** 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.

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

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

[10] **Hinze, M.** & **Walther, A.** An optimal memory-reduced procedure for calculating adjoints of the instationary Navier-Stokes equations. *MATH-NM-06-2002*, June 2002.

[11] **Glowinski, R.** Finite element methods for the numerical simulation of incompressible viscous flow. Introduction to the control of the Navier-Stokes equations. *Lectures in Applied Mathematics*, 28:219-301, 1991.

[12] **Kaltenbach, H.-J., Jürgens, W.,** & **Spille, A.** Numerische simulation, Beeinflussung und Eigenmoden-Analyze einer abgelösten Strömung mit Querkomponente. In *Ergebnisberichte SFB 557 TP A6, Technische Universität Berlin, 2000.*

[13] **Lehmann, U.** & **Walther, A.** The implementation and testing of time-minimal and resource-optimal parallel reversal schedules. In *Computational Science - ICCS 2002, Proceedings of the International Conference on Computational Science*, pages 1049 – 1058. Springer, 2002.

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

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

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

[17] **Walther, A.** & **Griewank, A.** 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.