Optimization Methods and Software
Vol. 22, No. 5, October 2007, 813–833

Taylor & Francis
Taylor & Francis Group

Efficient numerical solution of parabolic optimization problems by finite element methods

ROLAND BECKER*†, DOMINIK MEIDNER‡ and BORIS VEXLER§

†Laboratoire de Mathématiques Appliquées, Université de Pau et des Pays de l'Adour, IPRA–BP 1155, 64013 Pau Cedex, France
‡Institut für Angewandte Mathematik, Ruprecht-Karls-Universität Heidelberg, INF 294, 69120 Heidelberg, Germany
§Johann Radon Institute for Computational and Applied Mathematics (RICAM), Austrian Academy of Sciences, Altenberger Straße 69, 4040 Linz, Austria

(Received 18 February 2005; in final form 27 October 2006)

We present an approach for efficient numerical solution of optimization problems governed by parabolic partial differential equations. The main ingredients are: space-time finite element discretization, second-order optimization algorithms, and storage reduction techniques. We discuss the combination of these components for the solution of large-scale optimization problems.

Keywords: Optimal control; Parabolic equations; Finite elements; Newton's method; Storage reduction

AMS Subject Classification: 35K55; 35K90; 49K20; 49M05; 49M15; 49M29; 65N30

1. Introduction

In this paper, we discuss efficient numerical methods for solving optimization problems governed by parabolic partial differential equations. The optimization problems are formulated in a general setting including optimal control as well as parameter identification problems. Both, time and space discretization are based on the finite element method. This allows a natural translation of the optimality conditions from the continuous to the discrete level. For this type of discretizations, we present a systematic approach for precise computation of the derivatives required in optimization algorithms. The evaluation of these derivatives is based on the solutions of appropriate adjoint (dual) and sensitivity (tangent) equations.

The solution of the underlying state equation is typically required in the whole time interval for the computation of these additional solutions. If all data are stored, the storage grows linearly with respect to the number of time intervals in the time discretization. This makes the optimization procedure prohibitive for fine discretizations. We suggest an algorithm, which

*Corresponding author. Email: roland.becker@univ-pau.fr

Optimization Methods and Software
ISSN 1055-6788 print/ISSN 1029-4937 online © 2007 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/10556780701228532

allows to reduce the required storage. We analyze the complexity of this algorithm and prove that the required storage grows only logarithmic with respect to the number of time intervals. Such results are well known for gradient evaluations in the context of automatic differentiation, see [1,2]. However, to the authors' knowledge, the analysis of the required numerical effort for the whole optimization algorithm is new. The presented approach is an extension of windowing strategies introduced in ref. [3].

The main contribution of this paper is the combination of the exact computation of the derivatives based on the space-time finite element discretization with the storage reduction techniques.

Space-time finite element discretizations of parabolic optimization problems discussed in this paper allow for systematic *a posteriori* error estimation and efficient adaptive algorithms to be discussed in forthcomming paper [4].

In this paper, we consider optimization problems under constraints of (nonlinear) parabolic differential equations

$$\partial_t u + A(q, u) = f, \quad u(0) = u_0(q). \tag{1}$$

Here, the state variable is denoted by u and the control variable by q. Both, the differential operator A and the initial condition u_0 may depend on q. This allows a simultaneous treatment of both, optimal control and parameter identification problems. For optimal control problems, the operator A is typically given by $A(q, u) = \bar{A}(u) + B(q)$, with a (nonlinear) operator $\bar{A}$ and (usually linear) control operator B. In parameter identification problems, the variable q denotes the unknown parameters to be determined and may enter the operator A in a nonlinear way. The case of initial control is included via the q-dependent initial condition $u_0(q)$.

The target of the optimization is to minimize a given cost functional $J(q, u)$ subject to the state equation (1).

The paper is organized as follows: In the next section, we describe an abstract optimization problem with a parabolic state equation written in a weak form and discuss optimality conditions. Then, the problem is reformulated as an unconstrained (reduced) optimization problem and the expressions for the required derivatives are provided. After that, we describe Newton-type methods for solution of the problem on the continuous level. In section 3, we discuss the space and time discretizations. The space discretization is done by conforming finite elements and for the time discretization we use two approaches: discontinuous Galerkin (dG) and continuous Galerkin (cG) methods, see, e.g. [5]. For both, we provide techniques for precise evaluation of the derivatives in the corresponding discrete problems. This allows for simple translation of the optimization algorithm described in section 2 from the continuous to the discrete level. section 4 is devoted to the storage reduction techniques. Here, we present and analyze an algorithm, which we call multi-level windowing, allowing for drastically reduction of the required storage by the computation of adjoint solutions. This algorithm is then specified for the computation of the derivatives required in the optimization loop. In the last section, we present numerical results illustrating our approach.

2. Optimization

The optimization problems considered in this paper, are formulated in the following abstract setting: Let Q be a Hilbert space for the controls (parameters) with scalar product $(\cdot, \cdot)_Q$. Moreover, let V and H be Hilbert spaces, which build together with the dual space V^* a Gelfand triple: $V \hookrightarrow H \hookrightarrow V^*$. The duality pairing between the Hilbert space V and its dual V^* is denoted by $\langle \cdot, \cdot \rangle_{V^* \times V}$ and the scalar product in H by $(\cdot, \cdot)_H$.

Let, moreover, $I = (0, T)$ be a time interval and the space X be defined as

$$X = \{v | v \in L^2(I, V) \text{ and } \partial_t v \in L^2(I, V^*)\}. \tag{2}$$

It is well known, that X is continuously embedded in $C(\bar{I}, H)$, see, e.g. [6].

After these preliminaries, we pose the state equation in a weak form using the form $\bar{a}(\cdot, \cdot)(\cdot)$ defined on $Q \times V \times V$, which is assumed to be twice continuously differentiable and linear in the third argument. The state variable $u \in X$ is determined by

$$(\partial_t u, \varphi)_I + a(q, u)(\varphi) + (u(0) - u_0(q), \varphi(0))_H = (f, \varphi)_I \quad \forall \varphi \in X, \tag{3}$$

where $f \in L^2(0, T; V^*)$ represents the right-hand side of the state equation and $u_0 \colon Q \to H$ denotes a twice continuously differentiable mapping describing parameter-dependent initial conditions. For brevity of notation, we omit the arguments t and x of time-dependent functions whenever possible and use the abbreviations $(v, w)_I := \int_I (v, w)_H \, dt$ and $a(q, u)(\varphi) := \int_I \bar{a}(q, u)(\varphi) \, dt$. Note, that the inner product $(v, w)_H$ can be extended to $v \in V^*$, $w \in V$ by properties of the Gelfand triple.

The cost functional $J \colon Q \times X \to \mathbb{R}$ is defined using two twice continuously differentiable functionals $J_1 \colon V \to \mathbb{R}$ and $J_2 \colon H \to \mathbb{R}$ by:

$$J(q, u) = \int_I J_1(u) \, dt + J_2(u(T)) + \frac{\alpha}{2} \|q - \bar{q}\|_Q^2, \tag{4}$$

where the regularization (or cost) term involving $\alpha \geq 0$ and a reference parameter $\bar{q} \in Q$ is added.

The corresponding optimization problem is formulated as follows:

$$\text{Minimize } J(q, u) \text{ subject to (3)}, \quad (q, u) \in Q \times X. \tag{5}$$

The question of existence and uniqueness of solutions to such optimization problems is discussed in, e.g. [7–10]. Throughout the paper, we assume problem (5) to admit a (locally) unique solution.

Furthermore, we assume the existence of a twice continuously differentiable solution operator $S \colon Q \to X$ of the state equation (3). Using this solution operator we introduce the reduced cost functional $j \colon Q \to \mathbb{R}$, defined by $j(q) = J(q, S(q))$. This definition allows to reformulate problem (5) as an unconstrained optimization problem:

$$\text{Minimize } j(q), \quad q \in Q. \tag{6}$$

To express the first and second derivatives of the reduced cost functional j required in optimization algorithms, we introduce the Lagrangian $\mathcal{L} \colon Q \times X \times X \to \mathbb{R}$, defined as

$$\mathcal{L}(q, u, z) = J(q, u) + (f - \partial_t u, z)_I - a(q, u)(z) - (u(0) - u_0(q), z(0))_H. \tag{7}$$

With the help of the Lagrangian, we now present three auxiliary equations, which we will use in the sequel to give expressions of the derivatives of the reduced functional. Each equation will thereby be given in two formulations, first in terms of the Lagrangian and then using the concrete form of the optimization problem under consideration.

- *Dual equation*: For given $q \in Q$ and $u = S(q) \in X$, find $z \in X$ such that for all $\varphi \in X$

$$\mathcal{L}'_u(q, u, z)(\varphi) = 0,$$

$$-(\varphi, \partial_t z)_I + a'_u(q, u)(\varphi, z) + (z(T), \varphi(T))_H = \int_I J'_1(u)(\varphi) \, dt + J'_2(u(T))(\varphi(T)) \tag{8}$$

- *Tangent equation*: For given $q \in Q$, $u = S(q) \in X$, and a given direction $\delta q \in Q$, find $\delta u \in X$ such that for all $\varphi \in X$

$$\mathcal{L}''_{qz}(q,u,z)(\delta q,\varphi) + \mathcal{L}''_{uz}(q,u,z)(\delta u,\varphi) = 0,$$
$$(\partial_t \delta u,\varphi)_I + a'_u(q,u)(\delta u,\varphi) + (\delta u(0) - u'_0(q)(\delta q),\varphi(0))_H = -a'_q(q,u)(\delta q,\varphi). \quad (9)$$

- *Dual for Hessian equation*: For given $q \in Q$, $u = S(q) \in X$, $z \in X$ the corresponding solution of the dual equation (8), and $\delta u \in X$ the solution of the tangent equation (9) for the given direction δq, find $\delta z \in X$ such that for all $\varphi \in X$

$$\mathcal{L}''_{qu}(q,u,z)(\delta q,\varphi) + \mathcal{L}''_{uu}(q,u,z)(\delta u,\varphi) + \mathcal{L}''_{zu}(q,u,z)(\delta z,\varphi) = 0,$$
$$-(\varphi,\partial_t \delta z)_I + a'_u(q,u)(\varphi,\delta z) + (z(T),\varphi(T))_H = \int_I J''_1(u)(\delta u,\varphi)\,dt$$
$$+J''_2(u(T))(\delta u(T),\varphi(T)) - a''_{uu}(q,u)(\delta u,\varphi,z) - a''_{qu}(q,u)(\delta q,\varphi,z)\}. \quad (10)$$

To get the explicite representation (9) of the tangent equation, we only need to calculate the derivatives of the Lagrangian (7). The derivation of the explicite representations (8) and (10) of the dual and the dual for Hessian equation is done using additionally integration by parts in time.

In virtue of the dual equation defined above, we can now state an expression for the first derivatives of the reduced functional:

THEOREM 2.1 *Let for given $q \in Q$:*

(i) *$u = S(q) \in X$ be a solution of the state equation* (3).
(ii) *$z \in X$ fulfill the dual equation* (8).

Then there holds $j'(q)(\tau q) = \mathcal{L}'_q(q,u,z)(\tau q)$, which we may expand as

$$j'(q)(\tau q) = \alpha(q - \bar{q},\tau q)_Q - a'_q(q,u)(\tau q,z) + (u'_0(q)(\tau q),z(0))_H. \quad (11)$$

Proof Since condition (i) ensures that u is the solution of the state equation (3), and due to the definition (7) of the Lagrangian, we obtain: $j(q) = \mathcal{L}(q,u,z)$. By taking (total) derivative of the above equality with respect to q in direction τq, we get

$$j'(q)(\tau q) = \mathcal{L}'_q(q,u,z)(\tau q) + \mathcal{L}'_u(q,u,z)(\tau u) + \mathcal{L}'_z(q,u,z)(\tau z),$$

where $\tau u = S'(q)(\tau q)$, and $\tau z \in X$ is the derivatives of z with respect to q in direction τq. Noting the equivalence of condition (i) with $\mathcal{L}'_z(q,u,z)(\varphi) = 0$ for all $\varphi \in X$ and calculating the derivative of the Lagrangian (7) completes the proof. ■

To use Newton's method for solving the considered optimization problems, we have to compute the second derivatives of the reduced functional. The following theorem presents two alternatives for doing that. These two versions lead to two different optimization loops, which are presented in the sequel.

THEOREM 2.2 *Let for given $q \in Q$ the conditions of Theorem* 2.1 *be fulfilled.*

(a) *Moreover, let for given $\delta q \in Q$:*
  (i) *$\delta u \in X$ fulfill the tangent equation* (9).
  (ii) *$\delta z \in X$ fulfill the dual for Hessian equation* (10).

Then there holds for all $\tau q \in Q$

$$j''(q)(\delta q, \tau q) = \mathcal{L}''_{qq}(q,u,z)(\delta q, \tau q) + \mathcal{L}''_{uq}(q,u,z)(\delta u, \tau q) + \mathcal{L}''_{zq}(q,u,z)(\delta z, \tau q),$$

which we may equivalently express as

$$\begin{aligned} j''(q)(\delta q, \tau q) = \alpha(\delta q, \tau q)_Q - a''_{qq}(q,u)(\delta q, \tau q, z) - a''_{uq}(q,u)(\delta u, \tau q, z) \\ - a'_q(q,u)(\tau q, \delta z) + (u'_0(q)(\tau q), \delta z(0))_H + (u''_0(q)(\delta q, \tau q), z(0))_H. \end{aligned} \tag{12}$$

(b) *Moreover, let for given* $\delta q, \tau q \in Q$:
(i) $\delta u \in X$ *fulfill the tangent equation* (9) *for the given direction* δq.
(ii) $\tau u \in X$ *fulfill the tangent equation* (9) *for the given direction* τq.
Then there holds

$$\begin{aligned} j''(q)(\delta q, \tau q) = \mathcal{L}''_{qq}(q,u,z)(\delta q, \tau q) + \mathcal{L}''_{uq}(q,u,z)(\delta u, \tau q) \\ + \mathcal{L}''_{qu}(q,u,z)(\delta q, \tau u) + \mathcal{L}''_{uu}(q,u,z)(\delta u, \tau u), \end{aligned}$$

which we may equivalently express as

$$\begin{aligned} j''(q)(\delta q, \tau q) = \alpha(\delta q, \tau q) + \int_I J_1''(u)(\delta u, \tau u)\, dt - a''_{qq}(q,u)(\delta q, \tau q, z) \\ - a''_{uq}(q,u)(\delta u, \tau q, z) - a''_{qu}(q,u)(\delta q, \tau u, z) \\ - a''_{uu}(q,u)(\delta u, \tau u, z) + J_2''(u(T))(\delta u(T), \tau u(T)). \end{aligned} \tag{13}$$

Proof Due to condition (i) of Theorem 2.1, we obtain as before

$$j'(q)(\delta q) = \mathcal{L}'_q(q,u,z)(\delta q) + \mathcal{L}'_u(q,u,z)(\delta u) + \mathcal{L}'_z(q,u,z)(\delta z),$$

and taking (total) derivatives with respect to q in direction τq yields with the notations of the proof of Theorem 2.1 and additionally $\delta^2 u = S''(q)(\delta q, \tau q)$ and $\delta^2 z \in X$ the second derivative of z in the directions δq and τq

$$\begin{aligned} j''(q)(\delta q, \tau q) = \mathcal{L}''_{qq}()(\delta q, \tau q) + \mathcal{L}''_{qu}()(\delta q, \tau u) + \mathcal{L}''_{qz}()(\delta q, \tau z) + \mathcal{L}''_{uq}()(\delta u, \tau q) \\ + \mathcal{L}''_{uu}()(\delta u, \tau u) + \mathcal{L}''_{uz}()(\delta u, \tau z) + \mathcal{L}''_{zq}()(\delta z, \tau q) \\ + \mathcal{L}''_{zu}()(\delta z, \tau u) + \mathcal{L}'_u()(\delta \tau u) + \mathcal{L}'_z()(\delta \tau z). \end{aligned}$$

(For abbreviation we have omitted the content of the first parenthesis of the Lagrangian.)
We complete the proof applying the stated conditions to this expression. ■

In the sequel, we present two variants of the Newton-based optimization loop on the continuous level. The difference between these variants consists in the way of computing the update. Newton-type methods are used for solving optimization problem governed by time-dependent partial differential equations, see, e.g. [11,12].

From here on, we consider finite dimensional control space Q with a basis:

$$\{\tau q_i | i = 1, \ldots, \dim Q\}. \tag{14}$$

This is either due to the finite dimensional structure of the control space itself or due to its discretization.

Both, Algorithm 2.1 and Algorithm 2.3, describe an usual Newton-type method for the unconstrained optimization problem (6), which requires the solution of the following linear system in each iteration:

$$\nabla^2 j(q^n)\delta q = -\nabla j(q^n) \tag{15}$$

for the curent iterate q^n.

In both algorithms, the required gradient $\nabla j(q^n)$ is computed using representation (11) from Theorem 2.1. The algorithms differ in the way how they solve the linear system (15) to obtain a correction δq for the current control q. Algorithm 2.1 treats the computation of this system using the conjugate gradients method. It basically necessitates products of the Hessian with given vectors and does not need the entire Hessian.

ALGORITHM 2.1 *Optimization Loop without building up the Hessian*:

1: *Choose initial* $q^0 \in Q$ *and set* $n = 0$.
2: **repeat**
3: *Compute* $u^n \in X$, i.e. *solve the state equation* (3).
4: *Compute* $z^n \in X$, i.e. *solve the dual equation* (8).
5: *Build up the gradient* $\nabla j(q^n)$. *To compute its* i*-th component* $(\nabla j(q^n))_i$, *evaluate the right-hand side of representation* (11) *for* $\tau q = \tau q_i$.
6: *Solve* (15) *by use of the method of conjugate gradients. For the computation of the required matrix–vector products, apply the procedure described in Algorithm* 2.2.
7: *Set* $q^{n+1} = q^n + \delta q$.
8: *Increment* n.
9: **until** $\| \nabla j(q^n) \| < TOL$

The computation of the required matrix–vector products can be done with the representation given in Theorem 2.2(a) and is described in Algorithm 2.2. We note that in order to obtain the product of the Hessian with a given vector, we have to solve one tangent equation and one dual for Hessian equation. This has to be done in each step of the method of conjugate gradients.

ALGORITHM 2.2 *Computation of the matrix–vector product* $\nabla^2 j(q^n)\delta q$:

Require: u^n *and* z^n *are already computed for the given* q^n
1: *Compute* $\delta u^n \in X$, i.e. *solve the tangent equation* (9).
2: *Compute* $\delta z^n \in X$, i.e. *solve the dual for Hessian equation* (10).
3: *Build up the product* $\nabla^2 j(q^n)\delta q$. *To compute its* i*-th component* $(\nabla^2 j(q^n)\delta q)_i$, *evaluate the right-hand side of representation* (12) *for* $\tau q = \tau q_i$.

In contrast to Algorithm 2.1, Algorithm 2.3 builds up the whole Hessian. Consequently we may use every linear solver to the linear system (15). To compute the Hessian, we use the representation of the second derivatives of the reduced functional given in Theorem 2.2(b). Thus, in each Newton step we have to solve the tangent equation for each basis vector in (14).

ALGORITHM 2.3 *Optimization Loop with building up the Hessian*:

1: *Choose initial* $q^0 \in Q$ *and set* $n = 0$.
2: **repeat**
3: *Compute* $u^n \in X$, i.e. *solve the state equation* (3).
4: *Compute* $\{\tau u_i^n \mid i = 1, \ldots, \dim Q\} \subset X$ *for the chosen basis of* Q, i.e. *solve the tangent equation* (9) *for each of the basis vectors* τq_i *in* (14).

5: *Compute* $z^n \in X$, i.e. *solve the dual equation* (8).
6: *Build up the gradient* $\nabla j(q^n)$. *To compute its i-th component* $(\nabla j(q^n))_i$, *evaluate the right-hand side of representation* (11) *for* $\tau q = \tau q_i$.
7: *Build up the Hessian* $\nabla^2 j(q^n)$. *To compute its ij-th entry* $(\nabla^2 j(q^n))_{ij}$, *evaluate the right-hand side of representation* (13) *for* $\delta q = \tau q_j$, $\tau q = \tau q_i$, $\delta u = \tau u_j$, *and* $\tau u = \tau u_i$.
8: *Compute* δq *as the solution of* (15) *by use of an arbitrary linear solver.*
9: *Set* $q^{n+1} = q^n + \delta q$.
10: *Increment* n.
11: **until** $\| \nabla j(q^n) < TOL$

We now compare the efficiency of the two presented algorithms. For one step of Newton's method, Algorithm 2.1 requires the solution of two linear problems (tangent equation and dual for Hessian equation) for each step of the CG-iteration, whereas for Algorithm 2.3 it is necessary to solve dim Q tangent equations. Thus, if we have to perform n_{CG} steps of the method of conjugate gradients per Newton step, we should favor Algorithm 2.3, if

$$\frac{\dim Q}{2} \leq n_{\mathrm{CG}}. \tag{16}$$

In section 4, we will discuss a comparison of these two algorithms in the context of windowing.

Since n_{CG} can be bounded by the condition number of the reduced Hessian, one might estimate its size and dependance wit respect to dim Q for a concrete problem. In section 5 we present an example with small n_{CG}.

3. Discretization

In this section, we discuss the discretization of the optimization problem (5). To this end, we use finite element method in time and space to discretize the state equation. This allows us to give a natural computable representation of the discrete gradient and Hessian. The use of exact discrete derivatives is important for the convergence of the optimization algorithms.

We discuss the corresponding (discrete) formulation of the auxiliary problems (dual, tangent, and dual for Hessian) introduced in section 2. The first subsection is devoted to semi-discretization in time by continuous Galerkin (cG) and discontinuous Galerkin (dG) methods. Subsection 3.2 deals with the space discretization of the semi-discrete problems arising from the time discretization. We also present the form of the required auxiliary equations for one concrete realization of the cG and the dG discretization, respectively.

3.1 *Time discretization*

To define a semi-discretization in time, let us partition the time interval $\bar{I} = [0, T]$ as $\bar{I} = \{0\} \cup I_1 \cup I_2 \cup \cdots \cup I_M$ with subintervals $I_m = (t_{m-1}, t_m]$ of size k_m and time points $0 = t_0 < t_1 < \cdots < t_{M-1} < t_M = T$. We define the parameter k as a piecewise constant function by setting $k|_{I_m} = k_m$ for $m = 1, \ldots, M$.

3.1.1 Discontinuous Galerkin (dG) methods. We introduce for $r \in \mathbb{N}_0$ the discontinuous trial and test space

$$X_k^r = \{v_k \in L^2(I, V) | v_k|_{I_m} \in P_r(I_m, V),\ m = 1, \ldots, M,\ v_k(0) \in H\}. \tag{17}$$

Here, we denote $P_r(I_m, V)$ the space of polynomial of degree r defined on I_m with values in V. Additionally, we will use the following notations for functions $v_k \in X_k^r$:

$$v_{k,m}^+ = \lim_{t\to 0^+} v_k(t_m + t), \quad v_{k,m}^- = \lim_{t\to 0^+} v_k(t_m - t) = v_k(t_m), \quad [v_k]_m = v_{k,m}^+ - v_{k,m}^-.$$

The dG discretization of the state equation (3) now reads: Find $u_k \in X_k^r$ such that for all $\varphi \in X_k^r$ holds:

$$\sum_{m=1}^{M} (\partial_t u_k, \varphi)_{I_m} + a(q, u_k)(\varphi) + \sum_{m=1}^{M} ([u_k]_{m-1}, \varphi_{m-1}^+)_H + (u_{k,0}^- - u_0(q), \varphi(0))_H = (f, \varphi)_I \tag{18}$$

Here, we use the abbreviation $(v, w)_{I_m} := \int_{I_m} (v, w)_H dt$. For the analysis of the discontinuous finite element time discretization we refer to [5,13].

The corresponding semi-discrete optimization problem is given by:

$$\text{Minimize } J(q, u_k) \text{ subject to (18)}, \quad (q, u_k) \in Q \times X_k^r, \tag{19}$$

with the cost functional J from (4).

Similar to the continuous case, we introduce a semi-discrete solution operator $S_k \colon Q \to X_k^r$ such that $S_k(q)$ fulfills for $q \in Q$ the semi-discrete state equation (18). As in section 2, we define the semi-discrete reduced cost functional $j_k \colon Q \to \mathbb{R}$ as $j_k(q) = J(q, S_k(q))$, and reformulate the optimization problem (19) as unconstrained problem:

$$\text{Minimize } j_k(q), \quad q \in Q.$$

To derive a representation of the derivatives of j_k, we define the semi-discrete Lagrangian $\mathcal{L}_k \colon Q \times X_k^r \times X_k^r \to \mathbb{R}$, similar to the continuous case, as

$$\mathcal{L}_k(q, u_k, z_k) = J(q, u_k) + (f, z_k)_I - \sum_{m=1}^{M} (\partial_t u_k, z_k)_{I_m} - a(q, u_k)(z_k)$$
$$- \sum_{m=1}^{M} ([u_k]_{m-1}, z_{k,m-1}^+)_H - (u_{k,0}^- - u_0(q), z_{k,0}^-)_H.$$

With these preliminaries, we obtain similar expressions for the three auxiliary equations in terms of the semi-discrete Lagrangian as stated in the section before. However, the derivation of the explicite representations for the auxiliary equations requires some care due to the special form of the Lagrangian $\mathcal{L}_k$ for the dG discretization:

- *Dual equation for dG*: For given $q \in Q$ and $u_k = S_k(q) \in X_k^r$, find $z_k \in X_k^r$ such that

$$\sum_{m=1}^{M} -(\varphi, \partial_t z_k)_{I_m} + a_u'(q, u_k)(\varphi, z_k) - \sum_{m=1}^{M-1} (\varphi_m^-, [z_k]_m)_H$$
$$+ (\varphi_M^-, z_{k,M}^-)_H = \int_I J_1'(u_k)(\varphi)\, dt + J_2'(u_{k,M}^-)(\varphi_M^-), \quad \forall \varphi \in X_k^r. \tag{20}$$

- *Tangent equation for dG*: For $q \in Q$, $u_k = S_k(q) \in X_k^r$, and a given direction $\delta q \in Q$, find $\delta u_k \in X_k^r$ such that

$$\sum_{m=1}^{M} (\partial_t \delta u_k, \varphi)_{I_m} + a_u'(q, u_k)(\delta u_k, \varphi) + \sum_{m=1}^{M} ([\delta u_k]_{m-1}, \varphi_{m-1}^+)_H$$
$$+ (\delta u_{k,0}^-, \varphi_0^-)_H = -a_q'(q, u_k)(\delta q, \varphi) + (u_0'(q)(\delta q), \varphi_0^-)_H, \quad \forall \varphi \in X_k^r. \tag{21}$$

- *Dual for Hessian equation for dG*: For given $q \in Q$, $u_k = S_k(q) \in X_k^r$, $z_k \in X_k^r$ the corresponding solution of the dual equation (20), and $\delta u_k \in X_k^r$ the solution of the tangent equation (21) for the given direction δq, find $\delta z_k \in X_k^r$ such that

$$\begin{aligned}\sum_{m=1}^{M} -(\varphi, \partial_t \delta z_k)_{I_m} + a_u'(q, u_k)(\varphi, \delta z_k) - \sum_{m=1}^{M-1} (\varphi_m^-, [\delta z_k]_m)_H \\ + (\varphi_M^-, \delta z_{k,M}^-)_H = -a_{uu}''(q, u_k)(\delta u_k, \varphi, z_k) - a_{qu}''(q, u_k)(\delta q, \varphi, z_k) \\ + \int_I J_2''(u_k)(\delta u_k, \varphi)\, dt + J_2''(u_{k,M}^-)(\delta u_{k,M}^-, \varphi_M^-), \quad \forall \varphi \in X_k^r, \end{aligned} \tag{22}$$

As on the continuous level, the tangent equation can be obtained directly by calculating the derivatives of the Lagrangian, and for the dual equations, we additionally integrate by parts.

Now, the representations from Theorem 2.1 and Theorem 2.2 can be translated to the semi-discrete level: We obtain also on the semi-discrete level the representations (11)–(13) but now with the semi-discrete solutions u_k, δu_k, z_k, and δz_k instead of u, δu, z, and δz.

3.1.2 Continuous Galerkin (cG) methods. In this subsection, we discuss the time discretization by Galerkin methods with continuous trial functions and discontinuous test functions, the so called cG methods. For the test space, we use the space X_k^r defined in (17), and additionally, we introduce a trial space given by:

$$Y_k^s = \{v_k \in C(\bar{I}, H) | v_k|_{I_m} \in P_s(I_m, V),\ m = 1, \ldots, M\}.$$

To simplify the notation, we will use in this subsection the same symbols for the Lagrangian and the several solutions as in the subsection above for the dG discretization.

In virtue of these two spaces, we state the semi-discrete state equation in the cG context: Find $u_k \in Y_k^s$, such that

$$(\partial_t u_k, \varphi)_I + a(q, u_k)(\varphi) + (u_k(0) - u_0(q), \varphi(0))_H = (f, \varphi)_I, \quad \forall \varphi \in X_k^r. \tag{23}$$

Similarly to the previous subsection, we define the semi-discrete optimization problem

$$\text{Minimize } J(q, u_k) \text{ subject to (23)}, \quad (q, u_k) \in Q \times Y_k^s. \tag{24}$$

The Lagrangian $\mathcal{L}_k \colon Q \times Y_k^s \times X_k^r \to \mathbb{R}$ has the same form as on the continuous level, that is $\mathcal{L}_k(q, u_k, z_k) = \mathcal{L}(q, u_k, z_k)$.

Now, we can recall the process described in the previous subsection for the dG discretization to obtain the solution operator $S_k \colon Q \to Y_k^s$, the reduced functional j_k, and the unconstrained optimization problem.

For the cG discretization, the three auxiliary equations read as follows:

- *Dual equation for cG*: For given $q \in Q$ and $u_k = S_k(q) \in Y_k^s$, find $z_k \in X_k^r$ such that

$$\begin{aligned}\sum_{m=1}^{M} -(\varphi, \partial_t z_k)_{I_m} + a_u'(q, u_k)(\varphi, z_k) - \sum_{m=1}^{M-1} (\varphi(t_m), [z_k]_m)_H \\ + (\varphi(T), z_{k,M}^-)_H = \int_I J_1'(u_k)(\varphi)\, dt + J_2'(u_k(T))(\varphi(T)), \quad \forall \varphi \in Y_k^s. \end{aligned} \tag{25}$$

- *Tangent equation for cG*: For $q \in Q$, $u_k = S_k(q) \in Y_k^s$, and a given direction $\delta q \in Q$, find $\delta u_k \in Y_k^s$ such that

$$(\partial_t \delta u_k, \varphi)_I + a'_u(q, u_k)(\delta u_k, \varphi) + (\delta u_k(0), \varphi_0^-)_H = -a'_q(q, u_k)(\delta q, \varphi) + (u'_0(q), \varphi_0^-)_H, \quad \forall \varphi \in X_k^r. \tag{26}$$

- *Dual for Hessian equation for cG*: For given $q \in Q$, $u_k = S_k(q) \in Y_k^s$, $z_k \in X_k^r$ the corresponding solution of the dual equation (25), and $\delta u_k \in Y_k^s$ the solution of the tangent equation (26) for the given direction δq, find $\delta z_k \in X_k^r$ such that

$$\sum_{m=1}^{M} -(\varphi, \partial_t \delta z_k)_H + a'_u(q, u_k)(\varphi, \delta z_k) - \sum_{m=1}^{M-1} (\varphi(t_m), [\delta z_k]_m)_H + (\varphi(T), \delta z_{k,M}^-)_H = -a''_{uu}(q, u_k)(\delta u_k, \varphi, z_k) - a''_{qu}(q, u_k)(\delta q, \varphi, z_k) + \int_I J_1''(u_k)(\delta u_k, \varphi)\, dt + J_2''(u_k(T))(\delta u_k(T), \varphi(T)), \quad \forall \varphi \in Y_k^s. \tag{27}$$

The derivation of the tangent equation (26) is straightforward and similar to the continuous case. However, the dual equation (25) and the dual for Hessian equation (27) contain jump terms such as $[z_k]_m$ or $[\delta z_k]_m$ due to the interval-wise integration by parts.

Again, Theorem 2.1 and Theorem 2.2 are translated to the semi-discrete level by replacing the equations (8), (9) and (10) by the semi-discrete equations (25), (26) and (27). As already discussed for the dG discretization, the representations of the derivatives of j_k for the cG discretization have the same form as in the continuous case. Therefore, one should use formulas (11)–(13), where u, δu, z, and δz are replaced by u_k, δu_k, z_k, and δz_k which are determined by (23) and (25)–(27).

3.2 *Space-time discretization*

In this subsection, we first describe the finite element discretization in space. To this end, we consider two or three dimensional shape-regular meshes, see, e.g. [14]. A mesh consists of cells K, which constitute a nonoverlapping cover of the computational domain $\Omega \subset \mathbb{R}^d$, $d \in \{2, 3\}$. The corresponding mesh is denoted by $\mathcal{T}_h = \{K\}$, where we define the parameter h as a cellwise constant function by setting $h|_K = h_K$ with the diameter h_K of the cell K.

On the mesh $\mathcal{T}_h$ we construct a finite element space $V_h \subset V$ in standard way:

$$V_h = \{v \in V | v|_K \in \tilde{Q}_l(K) \text{ for } K \in \mathcal{T}_h\}.$$

Here, $\tilde{Q}_l(K)$ consists of shape functions obtained via (bi-)linear transformations of functions in $Q_l(\hat{K})$ defined on the reference cell $\hat{K} = (0, 1)^2$.

Now, the time-discretized schemes developed in the two previous subsections can be transfered to the full discretized level. For doing this, we use the spaces

$$X_{hk}^r = \{v_{hk} \in L^2(I, V_h) | v_{hk}|_{I_m} \in P_r(I_m, V_h),\ m = 1, \ldots, M,\ v_{hk}(0) \in V_h\}$$

and

$$Y_{hk}^s = \{v_{hk} \in C(\bar{I}, V_h) | v_{hk}|_{I_m} \in P_s(I_m, V_h),\ m = 1, \ldots, M\}$$

instead of X_k^r and Y_k^s.

Remark 3.1 Often, by solving problems with complex dynamical behavior, it is desirable to use time-dependent meshes $\mathcal{T}_{h_m}$. Then, h_m describes the mesh used in time interval I_m. Details of this construction are presented for the forward simulation of parabolic equations in ref. [15].

In the sequel, we present one concrete time-stepping scheme for the dG and the cG discretization combined with the finite element space discretization. These schemes correspond to the implicit Euler scheme and the Crank–Nicolson scheme, respectively.

To obtain the standard implicit Euler scheme as a special case of dG discretization, we choose $r = 0$ and approximate the integrals arising by the box rule. Furthermore, we define $U_m = u_{hk}|_{I_m}$, $\Delta U_m = \delta u_{hk}|_{I_m}$, $Z_m = z_{hk}|_{I_m}$, and $\Delta Z_m = z_{hk}|_{I_m}$ for $m = 1, \ldots, M$, and $U_0 = \bar{u}_{hk,0}$, $\Delta U_0 = \delta \bar{u}_{hk,0}$, $Z_0 = \bar{z}_{hk,0}$, and $\Delta Z_0 = \delta \bar{z}_{hk,0}$. With this, we obtain the following schemes for the dG-discretized state and auxiliary equations, which should be fulfilled for all $\varphi \in V_h$ and $m = 0, \ldots, M$.

- *State equation for dG*:

$$
\begin{aligned}
(U_0, \varphi)_H &= (u_0(q), \varphi)_H, \\
(U_m, \varphi)_H + k_m a(q, U_m)(\varphi) &= (U_{m-1}, \varphi)_H + k_m (f(t_m), \varphi)_H
\end{aligned}
$$

- *Dual equation for dG*:

$$
\begin{aligned}
(\varphi, Z_M)_H + k_M a'_u(q, U_M)(\varphi, Z_M) &= J'_2(U_M)(\varphi) + k_M J'_1(U_M)(\varphi), \\
(\varphi, Z_m)_H + k_m a'_u(q, U_m)(\varphi, Z_m) &= (\varphi, Z_{m+1})_H + k_m J'_1(U_m)(\varphi), \\
(\varphi, Z_0)_H &= (\varphi, Z_1)_H
\end{aligned}
$$

- *Tangent equation for dG*:

$$
\begin{aligned}
(\Delta U_0, \varphi)_H &= (u'_0(q)(\delta q), \varphi)_H, \\
\Delta U_m, \varphi)_H + k_m a'_u(q, U_m)(\Delta U_m, \varphi) &= (\Delta U_{m-1}, \varphi)_H - k_m a'_q(q, U_m)(\delta q, \varphi)
\end{aligned}
$$

- *Dual for Hessian equation for dG*:

$$
\begin{aligned}
(\varphi, \Delta Z_M)_H &+ k_M a'_u(q, U_M)(\varphi, \Delta Z_M) = J''_2(U_M)(\Delta U_M, \varphi) + k_M J''_1(U_M)(\Delta U_M, \varphi) \\
&- k_M \{a''_{uu}(q, U_M)(\Delta U_M, \varphi, Z_M) + a''_{qu}(q, U_M)(\delta q, \varphi, Z_M)\}, (\varphi, \Delta Z_m)_H \\
&+ k_m a'_u(q, U_m)(\varphi, \Delta Z_m) = (\varphi, \Delta Z_{m+1})_H + k_m J''_1(U_m)(\Delta U_m, \varphi) \\
&- k_m \{a''_{uu}(q, U_m)(\Delta U_m, \varphi, Z_m) + a''_{qu}(q, U_m)(\delta q, \varphi, Z_m)\}, (\varphi, \Delta Z_0)_H = (\varphi, \Delta Z_1)_H
\end{aligned}
$$

The Crank–Nicolson scheme can be obtained in the context of cG discretization by choosing $r = 0$, $s = 1$ and approximating the integrals arising by the trapezoidal rule. Using the representation of the Crank–Nicolson scheme as a cG-scheme, allows us directly to give a concrete form of the auxiliary equations leading to the exact computation of the discrete gradient and Hessian.

We set $U_m = u_{hk}(t_m)$ and $Z_m = z_{hk}|_{I_m}$ for $m = 1, \ldots, M$, and $U_0 = u_{hk}(0)$ and $Z_0 = \bar{z}_{hk,0}$. With this, we obtain the following schemes for the cG-discretized state and dual equations, which should be fulfilled for all $\varphi \in V_h$ and $m = 0, \ldots, M$:

- *State equation for cG*:

$$(U_0, \varphi)_H = (u_0(q), \varphi)_H,$$
$$(U_m, \varphi)_H + \frac{k_m}{2} a(q, U_m)(\varphi) = (U_{m-1}, \varphi)_H$$
$$-\frac{k_m}{2} a(q, U_{m-1})(\varphi) + \frac{k_m}{2} \{(f(t_{m-1}), \varphi)_H + (f(t_m), \varphi)_H\}$$

- *Dual equation for cG*:

$$(\varphi, Z_M)_H + \frac{k_M}{2} a'_u(q, U_M)(\varphi, Z_M) = J'_2(U_M)(\varphi) + \frac{k_M}{2} J'_1(U_M)(\varphi),$$
$$(\varphi, Z_m)_H + \frac{k_m}{2} a'_u(q, U_m)(\varphi, Z_m) = (\varphi, Z_{m+1})_H$$
$$-\frac{k_{m+1}}{2} a'_u(q, U_m)(\varphi, Z_{m+1}) + \frac{k_m + k_{m+1}}{2} J'_1(U_m)(\varphi),$$
$$(\varphi, Z_0)_H = (\varphi, Z_1)_H - \frac{k_1}{2} a'_u(q, U_0)(\varphi, Z_1) + \frac{k_1}{2} J'_1(U_0)(\varphi)$$

The schemes for the tangent and dual for Hessian equations have the same form as the state and the dual equations, respectively. We pass on the presentation here.

Remark 3.2 The structure of the time-steps for the dual and also the dual for Hessian equations is quite unusual. In the first and in the last steps, 'half-steps' occur, and in the other steps, terms containing the sizes of two neighboring time intervals k_m and k_{m+1} appear.

4. Storage reduction techniques

When computing the gradient of the reduced cost functional as described before, we need access to the solution of the state equation at all points in space and time while computing the dual equation. Similarly, we need the solution of the state, tangent, and dual equations for the solution of the dual for Hessian equation when computing matrix–vector products with the Hessian of the reduced functional. For large problems, especially in three dimensions, storing all the necessary data might be impossible. To overcome this difficulty, storage reduction techniques have been developed [1,3,16]. They have the property of reducing storage from $\mathcal{O}(M)$ to $\mathcal{O}(\log_2 M)$ at the cost of $\mathcal{O}(M \log_2 M)$ additional time steps. For the binomial checkpointing algorithm [1,16] optimal complexity has been proven.

The purpose of this section is to discuss storage reduction techniques in the context of the optimization algorithms described in section 2. In this paper, we restrict ourselves to the windowing technique, based on ideas from [3,17], which is also used for numerical examples in section 5.3.

4.1 *The abstract algorithm*

First, we consider the following abstract setting: Let two time-stepping schemes be given:

$$\mathrm{x}_{m-1} \longmapsto \mathrm{x}_m, \quad \text{for } m = 1, \ldots, M,$$
$$(\mathrm{y}_{m+1}, \mathrm{x}_m) \longmapsto \mathrm{y}_m, \quad \text{for } m = M-1, \ldots, 0,$$

together with a given initial value x_0 and the mapping $\mathrm{x}_M \mapsto \mathrm{y}_M$. All time-stepping schemes given for dG and cG discretization in the previous section are concrete realizations of these abstract schemes.

Additionally, we assume that the solutions $\mathtt{x}_m$ as well as $\mathtt{y}_m$ require the same amount of storage for all $m = 0, \ldots, M$. However, if this is not the case, the windowing technique presented in the sequel can be applied to clusters of time steps similar in size instead of single time steps. Such clustering is, for example, important by using dynamical meshes, since in this case, the amount of storage for a solution $\mathtt{x}_m$ depends on the current mesh.

The trivial approach to perform the forward and backwards iterations is to compute and store the whole forward solution $\{\mathtt{x}_m\}_{m=0}^{M}$, and use these values to compute the backwards solution $\{\mathtt{y}_m\}_{m=0}^{M}$. The required amount of storage S_0 in terms of the size of one forward solution $\mathtt{x}_m$ to do this is $S_0 = M + 1$. The number of forward steps W_0 necessary to compute the whole backwards solution is $W_0 = M$.

The aim of the following windowing algorithms is to reduce the needed storage by performing some additional forward steps. To introduce the windowing, we additionally assume that we can factorize the number of given time steps M as $M = PR$ with positive integers P and R. With this, we can separate the set of time points $\{0, \ldots, M\}$ in P slices each containing $R - 1$ time steps and $P + 1$ sets containing one element as

$$\{0, \ldots, M\} = \{0\} \cup \{1, \ldots, R-1\} \cup \{R\} \cup \cdots \\ \cdots \cup \{(P-1)R\} \cup \{(P-1)R + 1, \ldots, PR - 1\} \cup \{PR\}.$$

The algorithm now works as follows: First, we compute the forward solution $\mathtt{x}_m$ for $m = 1, \ldots, M$ and store the $P + 1$ samples $\{\mathtt{x}_{Rl}\}_{l=0}^{P}$. Additionally, we store the $R - 1$ values of $\mathtt{x}$ in the last slice. Now, we have the necessary information on $\mathtt{x}$ to compute $\mathtt{y}_m$ for $m = M, \ldots, (P-1)R + 1$. Thus, the values of $\mathtt{x}$ in the last slice are not longer needed. We can replace them with the values of $\mathtt{x}$ in the next-last slice, which we can directly compute using the time-stepping scheme since we stored the value $\mathtt{x}_{(P-2)R}$ in the first run. Thereby, we can compute $\mathtt{y}_m$ for $m = (P-1)R, \ldots, (P-2)R + 1$. This can now be done iteratively till we have computed $\mathtt{y}$ in the first slice and finally obtain the value $\mathtt{y}_0$. This so called One-Level Windowing is presented in detail in Algorithm 4.1.

ALGORITHM 4.1 ONELEVELWINDOWING(P, R, M):

Require: $M = PR$.
1: *Store* $\mathtt{x}_0$.
2: *Take* $\mathtt{x}_0$ *as initial value for* $\mathtt{x}$.
3: **for** $m = 1$ **to** $(P-1)R$ **do**
4: *Compute* $\mathtt{x}_m$.
5: **if** m *is a multiple of* R **then**
6: *Store* $\mathtt{x}_m$.
7: **for** $n = (P-1)R$ **downto** 0 **step** R **do**
8: *Take* $\mathtt{x}_n$ *as initial value for* $\mathtt{x}$.
9: **for** $m = n + 1$ **to** $n + R - 1$ **do**
10: *Compute* $\mathtt{x}_m$ *and store* $\mathtt{x}_m$.
11: **if** $n = M - R$ **then**
12: *Compute* $\mathtt{x}_M$ *and store* $\mathtt{x}_M$.
13: **for** $m = n + R$ **downto** $n + 1$ **do**
14: *Compute* $\mathtt{y}_m$ *in virtue of* $\mathtt{x}_m$ *and delete* $\mathtt{x}_m$ *from memory.*
15: **if** $n = 0$ **then**
16: *Compute* $\mathtt{y}_0$ *and delete* $\mathtt{x}_0$ *from memory.*

During the execution of Algorithm 4.1, the needed amount of memory is not exceeding $(P + 1) + (R - 1)$ forward solutions. Each of the $\mathtt{y}_m$'s is computed exactly once, so we need

M solving steps to obtain the whole solution $\mathtt{y}$. To compute the necessary values of $\mathtt{x}_m$, we have to solve $M + (P-1)(R-1)$ forward steps, since we have to compute each of the values of $\mathtt{x}$ in the first $P-1$ slices. We summarize:

$$S_1(P, R) = P + R, \quad W_1(P, R) = 2M - P - R + 1,$$

where again S_1 denotes the required amount of memory in terms of the size of one forward solution and W_1 the number of time steps to provide the forward solution $\mathtt{x}$ needed to compute the whole backwards solution $\mathtt{y}$.

Here, the subscript 1 suggests that we can extend this approach to factorizations of M in $L+1$ factors for $L \in \mathbb{N}$. This extension can be obtained via the following inductive argumentation: Assuming $M = M_0 M_1 \cdots M_L$ with positive integers M_l, we can apply the algorithm described above to the factorization $M = PR$ with $P = M_0$ and $R = M_1 M_2 \cdots M_L$, and then recursively to each of the P slices. This so called multi-level windowing is described in Algorithm 4.2. It has to be started with the call MULTILEVELWINDOWING$(0, 0, L, M_0, M_1, \ldots, M_L, M)$. Of course, there holds by construction ONELEVELWINDOWING(P, R, M) = MULTILEVELWINDOWING$(0, 0, 1, P, R, M)$

ALGORITHM 4.2 MULTILEVELWINDOWING$(s, l, L, M_0, M_1, \ldots, M_L, M)$:

Require $M = M_0 M_1 \cdots M_L$.
Set $P = M_l$ *and* $R = M_{l+1} \cdots M_L$.
if $l = 0$ **and** $s = 0$ **then**
  Store $\mathtt{x}_0$.
Take $\mathtt{x}_s$ *as initial value for* $\mathtt{x}$.
for $m = 1$ **to** $(P-1)R$ **do**
  Compute $\mathtt{x}_{s+m}$.
  if m *is a multiple of* R **then**
    Store $\mathtt{x}_{s+m}$.
for $n = (P-1)R$ **downto** 0 **step** R **do**
  if $l + 1 < L$ **then**
    Call MULTILEVELWINDOWING $(s + n, l + 1, L, M_0, M_1, \ldots, M_L, M)$.
  else
    Take $\mathtt{x}_{s+n}$ *as initial value for* $\mathtt{x}$.
    for $m = n + 1$ **to** $n + R - 1$ **do**
      Compute $\mathtt{x}_{s+m}$ *and store* $\mathtt{x}_{s+m}$.
    if $s + n = M - R$
      Compute $\mathtt{x}_M$ *and store* $\mathtt{x}_M$.
    for $m = n + R$ **downto** $n + 1$ **do**
      Compute $\mathtt{y}_{s+m}$ *in virtue of* $\mathtt{x}_{s+m}$ *and delete* $\mathtt{x}_{s+m}$ *from memory.*
    if $s + n = 0$ **then**
      Compute $\mathtt{y}_0$ *and delete* $\mathtt{x}_0$ *from memory.*

In the following theorem, we calculate the necessary amount of storage and the number of needed forward steps to perform the multi-level windowing described in Algorithm 4.2 for a given factorization $M = M_0 M_1 \cdots M_L$ of length $L + 1$:

THEOREM 4.1 *For given* $L \in \mathbb{N}_0$ *and a factorization of the number of time steps* M *as* $M = M_0 M_1 \cdots M_L$ *with* $M_l \in \mathbb{N}$, *the required amount of memory in the multi-level windowing to*

perform all backwards solution steps is

$$S_L(M_0, M_1, \ldots, M_L) = \sum_{l=0}^{L} (M_l - 1) + 2.$$

To achieve this storage reduction, the number of performed forward steps enhances to

$$W_L(M_0, M_1, \ldots, M_L) = (L+1)M - \sum_{l=0}^{L} \frac{M}{M_l} + 1.$$

Proof We prove the theorem by mathematical induction: In the case $L = 0$, we then have $S_0(M) = M + 1$ and $W_0(M) = M$. For the step $L - 1 \rightsquigarrow L$, we consider the factorization $M = M_0 M_1 \cdots M_{L-2}(M_{L-1} M_L)$ of length L additionally to the given one of length $L + 1$. Then we obtain

$$\begin{aligned} S_L(M_0, M_1, \ldots, M_{L-1}, M_L) = {} & S_{L-1}(M_0, M_1, \ldots, M_{L-1} M_L) \\ & - (M_{L-1} M_L - 1) + (M_{L-1} - 1) + (M_L - 1). \end{aligned}$$

In virtue of the induction hypothesis for S_{L-1}, it follows the assertion for S_L. Now, we prove the assertion for W_L. For this, we obtain after some manipulation the identity

$$\begin{aligned} W_L(M_0, M_1, \ldots, M_{L-1}, M_L) = {} & W_{L-1}(M_0, M_1, \ldots, M_{L-1} M_L) \\ & + \frac{M}{M_{L-1} M_L}(M_{L-1} - 1)(M_L - 1). \end{aligned}$$

This implies the assertion for W_L in virtue of the induction hypothesis for W_{L-1}. ■

If $M^{(1/L+1)} \in \mathbb{N}$, the minimum of $\widetilde{S}_L$ of all possible factorizations of length $L + 1$ is

$$\widetilde{S}_L = S_L(M^{(1/L+1)}, \ldots, M^{(1/L+1)}) = (L+1)(M^{(1/L+1)} - 1) + 2.$$

The numbers of forward steps for the memory-optimal factorization then results in

$$\widetilde{W}_L = W_L(M^{(1/L+1)}, \ldots, M^{(1/L+1)}) = (L+1)(M - M^{(L/L+1)}) + 1.$$

If we choose $L \approx \log_2 M$, then we obtain for the optimal factorization from above logarithmic growth of the necessary amount of storage:

$$\widetilde{S}_L = \mathcal{O}(\log_2 M), \quad \widetilde{W}_L = \mathcal{O}(M \log_2 M).$$

4.2 *Application to optimization*

In this subsection, we consider the multi-level windowing, described in the previous subsection, in the context of nonstationary optimization. We give a detailed estimate of the number of steps and the amount of memory required to perform one Newton step for a given number of levels $L \in \mathbb{N}$. For brevity, we will just write W_L and S_L instead of $W_L(M_0, M_1, \ldots, M_L)$ and $S_L(M_0, M_1, \ldots, M_L)$.

4.2.1 Optimization loop *without* building up the Hessian. First, we treat the variant of the optimization algorithms, which does not build up the entire Hessian of the reduced functional and is given in Algorithm 2.1. As stated in this algorithm, it is necessary to compute the value of the reduced functional and the gradient one time per Newton step. To apply the derived windowing techniques, we set $\mathrm{x} = u$, $\mathrm{y} = z$ and note that Algorithm 4.2 can easily be extended to compute the necessary terms for evaluating the functional and the gradient during the forward or backwards computation, respectively. Thus, the total number of time steps needed to do this, is $W^{\mathrm{grad}} = W_L + M$. The required amount of memory is $S^{\mathrm{grad}} = S_L$.

Additionally to the gradient, we need to compute one matrix–vector product of the Hessian times a given vector in each of the n_{CG} steps of the conjugate gradient method. This is done as described in Algorithm 2.2. For avoiding the storage of u or z in all time steps, we have to compute u, δu, z, and δz again in every CG step. Consequently, we set here $\mathrm{x} = (u, \delta u)$ and $\mathrm{y} = (z, \delta z)$. We obtain $W^{\mathrm{hess}} = 2(W_L + M)$ and $S^{\mathrm{hess}} = 2S_L$.

In total, we achieve

$$W^{(1)} = (1 + 2n_{\mathrm{CG}})(W_L + M) \quad \text{and} \quad S^{(1)} = \max(S^{\mathrm{grad}}, S^{\mathrm{hess}}) = 2S_L.$$

4.2.2 Optimization loop with building up the Hessian. For using Algorithm 2.3, it is necessary to compute u, δu_i $(i = 1, \ldots, \dim Q)$, and z. Again, the evaluation of the reduced functional is done during the first forward computation, and the evaluation of the gradient and the Hessian is done during the computation of z. So, we set $\mathrm{x} = (u, \delta u_1, \delta u_2, \ldots, \delta u_{\dim Q})$ and $\mathrm{y} = z$. The required number of steps and the needed amount of memory are

$$W^{(2)} = (1 + \dim Q)W_L + M \quad \text{and} \quad S^{(2)} = (1 + \dim Q)S_L.$$

4.2.3 Comparison of the two variants of the optimization algorithm. For $\dim Q \geq 1$, we obtain directly $S^{(2)} \geq S^{(1)}$. The relation between $W^{(1)}$ and $W^{(2)}$ depends on the factorization of M. A simple calculation leads to the following condition:

$$W^{(2)} \leq W^{(1)} \Longleftrightarrow \frac{\dim Q}{2} \leq n_{\mathrm{CG}} \left(1 + \frac{M}{W_L}\right).$$

If we choose L such that $W_L \approx M \log_2 M$, we can express the condition above just in terms of M as

$$W^{(2)} \lesssim W^{(1)} \Longleftrightarrow \frac{\dim Q}{2} \lesssim n_{\mathrm{CG}} \left(1 + \frac{1}{\log_2 M}\right).$$

This means, that even though the required memory for the second algorithm with building up the Hessian is greater, this algorithm needs only fewer steps than the first one, if the necessary numbers of CG steps performed in each Newton step is greater than half of the dimension of Q times a factor depending logarithmic on the number of time steps M.

5. Numerical results

Here, we present illustrative numerical examples. Throughout the spatial discretization is done with piecewise bilinear/trilinear finite elements on quadrilateral or hexahedral cells. The resulting nonlinear state equations are solved by Newton's method, whereas the linear subproblems are treated by a multigrid method. For time discretization, we consider the variants of the cG and dG methods which we have presented in section 3. In the subsections 5.1 and

5.2, we only present results using the variant of the optimization loop building up the entire Hessian, described in Algorithm 2.3 since the results of the other variant are mainly the same.

All computations are done with the software packages RODOBO [18] and GASCOIGNE [19]. To depict the computed solutions, the visualization software VISUSIMPLE [20] was used.

We consider the following two example problems on the space-time domain $\Omega \times (0, T)$ with $T = 1$.

- *Example 1* In the first example, we discuss an optimal control problem with terminal observation, where the control variable enters the initial condition of the (nonlinear) state equation. We choose $\Omega = (0, 1)^3 \subset \mathbb{R}^3$ and pose the state equation as

$$\begin{aligned} \partial_t u - \nu \Delta u + u^2 = 0 \text{ in } \Omega \times (0, T), \quad \partial_n u = 0 \text{ on } \partial\Omega \times (0, T), \\ u(0, \cdot) = g_0 + \sum_{i=1}^{8} g_i q_i \text{ on } \Omega, \end{aligned} \tag{28}$$

  where $\nu = 0.1$, $g_0 = (1 - 2\|x - \bar{x}_0\|)^{30}$ with $\bar{x}_0 = (0.5, 0.5, 0.5)^T$ and $g_i = (1 - 0.5\|x - \bar{x}_i\|)^{30}$ with $\bar{x}_i \in \{0.2, 0.8\}^3$ for $i = 1, \ldots, 8$ are given.

  For an additionally given reference solution $\bar{u}_T(x) = (1/6)(3 + x_1 + x_2 + x_3)$, $x = (x_1, x_2, x_3)^T$, the optimization problem now reads as

$$\text{Minimize } \frac{1}{2} \int_\Omega (u(T, \cdot) - \bar{u}_T)^2 \, dx + \frac{\alpha}{2} \|q\|_Q^2 \text{ subject to (28)}, \quad (q, u) \in Q \times X,$$

  where $Q = \mathbb{R}^8$ and X is chosen in virtue of (2) with $V = H^1(\Omega)$ and $H = L^2(\Omega)$. The regularization parameter α is set to 10^{-4}.
- *Example 2* In the second example, we choose $\Omega = (0, 1)^2 \subset \mathbb{R}^2$ and consider a parameter estimation problem with the state equation given by

$$\begin{aligned} \partial_t u - \nu \Delta u + q_1 \partial_1 u + q_2 \partial_2 u = 2 + \sin(10\pi t) \quad \text{in } \Omega \times (0, T), \\ u = 0 \quad \text{on } \partial\Omega \times (0, T), \quad u(0, \cdot) = 0 \quad \text{on } \Omega, \end{aligned} \tag{29}$$

  where we again set $\nu = 0.1$.

  We assume to be given measurements $\bar{u}_{T,1}, \ldots, \bar{u}_{T,5} \in \mathbb{R}$ of the point values $u(T, p_i)$ for five different measurement points $p_i \in \Omega$. The unknown parameters $(q_1, q_2) \in Q = \mathbb{R}^2$ are estimated using a least squares approach resulting in the following optimization problem:

$$\text{Minimize } \frac{1}{2} \sum_{i=1}^{5} (u(T, p_i) - \bar{u}_{T,i})^2 \text{ subject to (29)}, \quad (q, u) \in Q \times X.$$

The consideration of point measurements does not fulfill the assumption on the cost functional in (4), since the point evaluation is not bounded as a functional on $H = L^2(\Omega)$. Therefore, the point functionals here may be understood as regularized functionals defined on $L^2(\Omega)$. For an *a priori* error analysis of an elliptic parameter identification problems with pointwise measurements we refer to [21].

5.1 *Validation of the computation of derivatives*

To verify the computation of the gradient ∇j_{hk} and the Hessian $\nabla^2 j_{hk}$ of the reduced cost functional, we consider the first and second difference quotients

$$\frac{j_{hk}(q+\epsilon\delta q) - j_{hk}(q-\epsilon\delta q)}{2\epsilon} = (\nabla j_{hk}, \delta q) + e_1,$$
$$\frac{j_{hk}(q+\epsilon\delta q) - 2j_{hk}(q) + j_{hk}(q-\epsilon\delta q)}{\epsilon^2} = (\delta q, \nabla^2 j_{hk}\delta q) + e_2.$$

By standard convergence and stability analysis we have

$$e_1 \approx c_1\epsilon^2\nabla^3 j_{hk}(\xi_1) + c_2\epsilon^{-1}, \quad e_2 \approx c_3\epsilon^2\nabla^4 j_{hk}(\xi_2) + c_4\epsilon^{-2},$$

where $\xi_1, \xi_2 \in (q-\epsilon\delta q, q+\epsilon\delta q)$ are intermediate points and the constants c_i do not depend on ϵ.

The tables 1 and 2 show the errors between the values of the derivatives computed by use of the difference quotients above and by use of the approach presented in the sections 2 and 3, for the considered examples. The values of these errors and the orders of convergence of the reduction of these errors for $\epsilon \to 0$ are given in the tables 1 and 2. Note, that the values of the derivatives computed via the approach based on the ideas presented in section 2 do not depend on ϵ.

The content of these tables does not considerably depend on the discretization parameters h and k, so we have the exact discrete derivatives also on coarse meshes or when using large time steps.

Table 1. Convergence of the difference quotients for the gradient and the Hessian of the reduced cost functional for Example 1 with $q = (0, \ldots, 0)^T$ and $\delta q = (1, \ldots, 1)^T$.

	Discontinuous Galerkin				Continuous Galerkin			
	Gradient		Hessian		Gradient		Hessian	
ϵ	e_1	Conv.	e_2	Conv.	e_1	Conv.	e_2	Conv.
1.0e-00	8.56e-01	–	6.72e-01	–	7.96e-01	–	5.97e-01	–
1.0e-01	5.37e-03	2.20	4.32e-03	2.19	5.28e-03	2.17	4.08e-03	2.16
1.0e-02	5.35e-05	2.00	4.27e-05	2.00	5.26e-05	2.00	4.05e-05	2.00
1.0e-03	5.34e-07	2.00	3.28e-05	0.11	5.26e-07	2.00	3.27e-05	0.09
1.0e-04	5.30e-09	2.00	8.49e-05	−0.41	5.41e-09	1.98	8.47e-05	−0.41
1.0e-05	2.91e-10	1.25	9.16e-05	−0.03	3.24e-10	1.22	7.25e-05	0.06

Table 2. Convergence of the difference quotients for the gradient and the Hessian of the reduced cost functional for Example 2 with $q = (6, 6)^T$ and $\delta q = (1, 1)^T$.

	Discontinuous Galerkin				Continuous Galerkin			
	Gradient		Hessian		Gradient		Hessian	
ϵ	e_1	Conv.	e_2	Conv.	e_1	Conv.	e_2	Conv.
1.0e-00	1.44e-01	–	8.09e-02	–	2.80e-01	–	1.33e-01	–
1.0e-01	1.36e-03	2.02	7.76e-04	2.01	2.59e-03	2.03	1.27e-03	2.02
1.0e-02	1.36e-05	2.00	7.75e-06	2.00	2.59e-05	2.00	1.27e-05	2.00
1.0e-03	1.36e-07	1.99	4.32e-07	1.25	2.59e-07	1.99	3.97e-07	1.50
1.0e-04	2.86e-09	1.67	5.01e-05	−2.06	2.83e-09	1.96	5.56e-06	−1.14
1.0e-05	5.94e-08	−1.31	2.18e-02	−2.63	9.95e-08	−1.54	2.00e-02	−3.55

5.2 *Optimization*

In this subsection, we apply the two optimization algorithms described in section 2 to the our optimization problems. For both examples, we present the results for both time discretization schemes presented in section 3.

In tables 3 and 4, we show the progression of the norm of the gradient of the reduced functional $\|\nabla j_{hk2}\|$ and the reduction of the cost functional j_{hk} during the Newton iteration for Example 1 and Example 2, respectively.

The computations for Example 1 were done on a mesh consisting of 4096 hexahedral cells with diameter $h = 0.0625$. The time interval $(0, 1)$ is split into 100 slices of size $k = 0.01$. The results of these computations are depicted in figure 1.

For Example 2, we chose a quadrilateral mesh with mesh size $h = 0.03125$ consisting of 1024 cells. The size of the time steps was set as $k = 0.005$ corresponding to 200 time steps. In table 4, we additionally show the value of the estimated parameters during the optimization run. The values of 'measurements' are taken from a solution of the state equation on a fine mesh consisting of 65536 cells with 5000 time steps for the 'exact' values of parameters chosen as $q_{\text{exact}} = (7, 9)^T$.

We note that due to condition (16), for Example 1 the variant of the optimization algorithm, which only uses matrix–vector products of the Hessian is the more efficient one, whereas for Example 2 one should use the variant which builds up the entire Hessian.

Table 3. Results of the optimization loop with dG and cG discretization for Example 1 starting with initial guess $q_0 = (0, \ldots, 0)^T$.

	Discontinuous Galerkin			Continuous Galerkin		
Step	n_{CG}	$\|\nabla_{j_{hk}}\|$	j_{hk}	n_{CG}	$\|\nabla_{j_{hk}}\|$	j_{hk}
0	–	1.21e-01	2.76e-01	–	1.21e-01	2.76e-01
1	2	4.99e-02	1.34e-01	2	4.98e-02	1.34e-01
2	2	2.00e-02	6.28e-02	2	1.99e-02	6.33e-02
3	3	7.61e-03	2.94e-02	3	7.62e-03	3.00e-02
4	3	2.55e-03	1.64e-02	3	2.57e-03	1.70e-02
5	3	6.03e-04	1.32e-02	3	6.21e-04	1.37e-02
6	3	5.72e-05	1.29e-02	3	6.18e-05	1.34e-02
7	3	6.37e-07	1.29e-02	3	7.62e-07	1.34e-02
8	3	1.75e-10	1.29e-02	3	1.21e-10	1.34e-02

Table 4. Results of the optimization loop with dG and cG discretization for Example 2.

	Discontinuous Galerkin				Continuous Galerkin			
Step	n_{CG}	$\|\nabla j_{hk}\|_2$	j_{hk}	q	n_{CG}	$\|\nabla j_{hk}\|_2$	j_{hk}	q
0	–	1.54e-02	1.73e-03	$(6.00, 6.00)^T$	–	1.25e-02	1.23e-03	$(6.00, 6.00)^T$
1	2	5.37e-04	4.53e-04	$(5.97, 7.72)^T$	2	4.35e-04	3.07e-03	$(6.06, 7.43)^T$
2	2	1.65e-04	7.85e-05	$(6.80, 8.52)^T$	2	1.29e-04	4.75e-05	$(6.48, 8.37)^T$
3	2	3.44e-05	5.56e-06	$(7.18, 9.19)^T$	2	2.48e-05	2.35e-06	$(6.87, 8.84)^T$
4	2	2.54e-06	9.20e-07	$(7.35, 9.39)^T$	2	1.47e-06	9.29e-09	$(6.99, 8.98)^T$
5	2	1.66e-08	8.91e-07	$(7.36, 9.41)^T$	2	6.10e-09	2.04e-10	$(6.99, 8.99)^T$
6	2	7.35e-13	8.91e-07	$(7.36, 9.41)^T$	1	5.89e-11	2.04e-10	$(6.99, 8.99)^T$

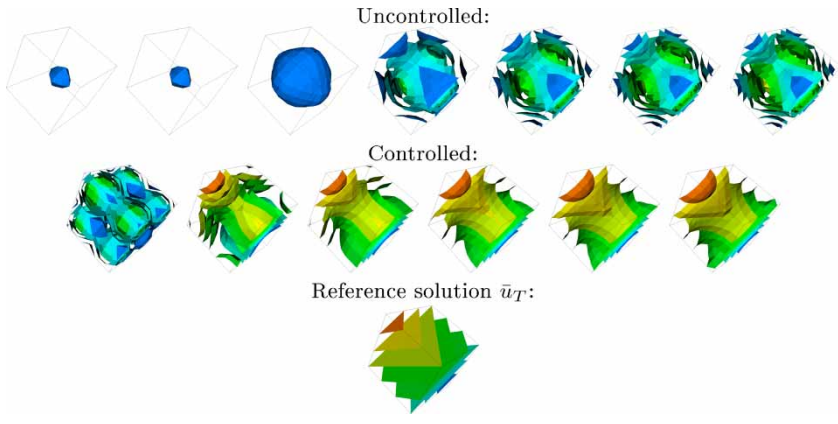

Figure 1. Iso-surfaces of the state of example problem 1 for time $t \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ before and after optimization.

5.3 *Windowing*

This subsection is devoted to the practical verification of the presented multi-level windowing. For this, we consider Example 1 with dG time discretization on a grid consisting of 32768 cells performing 500 time steps. Table 5 demonstrates the reduction of the storage requirement described in section 4. We can achieve a storage reduction about the factor 30 for both variants of the optimization loop. Thereby total number of steps only grows about the factor 3.2 for the algorithm with, and 4.0 for the algorithm without building up the entire Hessian.

We remark that although the factorization $2 \cdot 2 \cdot 5 \cdot 25$ consists of more factors than the factorization $5 \cdot 10 \cdot 10$, both, the storage requirement and the total number of time steps are greater for first factorization than for the second one. The reason for this is the imbalance of the size of the different factors in $2 \cdot 2 \cdot 5 \cdot 25$. As showed in section 4, in the optimal factorization are all factors the same. So, it is evident, that a factorization as $5 \cdot 10 \cdot 10$ is more efficient than one where the size factors varies very much.

Table 5 also proves the asserted dependence of the condition when to use which variant of the optimization loop on the considered factorization on M. For the factorizations $5 \cdot 100$ and $10 \cdot 50$ the variant with building up the Hessian needs less forward steps than the other variant without building up the Hessian. However, for the remaining factorizations the situation is the opposite way around.

Table 5. Reduction of the storage requirement due to Windowing in Example 1 with dG discretization and 32768 cells in each time step.

	With Hessian		Without Hessian	
Factorization	Memory in MB	Time steps	Memory in MB	Time steps
500	1236	45000	274	35000
$5 \cdot 100$	259	80640	58	87948
$10 \cdot 50$	148	84690	32	90783
$2 \cdot 2 \cdot 5 \cdot 25$	78	120582	17	118503
$5 \cdot 10 \cdot 10$	59	114174	13	113463
$4 \cdot 5 \cdot 5 \cdot 5$	41	136512	9	130788
$2 \cdot 2 \cdot 5 \cdot 5 \cdot 5$	39	146646	9	138663

References

[1] Griewank, A., 1992, Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, **1**, 35–54.

[2] Griewank, A., 2000, Evaluating derivatives, principles and techniques of algorithmic differentiation. *Frontiers in Applied Mathematics*, Vol. 19, (Philadelphia: SIAM).

[3] Berggren, M., Glowinski, R. and Lions, J.-L., 1996, A computational approach to controllability issues for flow-related models. (I): Pointwise control of the viscous burgers equation. *International Journal of Computational Fluid Dynamics*, **7**, 237–253.

[4] Meidner, D. and Vexler, B., 2006, Adaptive space-time finite element methods for parabolic optimization problems. *SIAM Journal on Control and Optimization*, to appear.

[5] Eriksson, K., Johnson, C. and Thomée, V., 1985, Time discretization of parabolic problems by the discontinuous Galerkin method. *RAIRO Modelisation Mathematical Analysis and Numerical Methods*, **19**, 611–643.

[6] Dautray, R. and Lions, J.-L., 1992, *Mathematical Analysis and Numerical Methods for Science and Technology: Evolution Problems I*, Vol. 5 (Berlin: Springer-Verlag).

[7] Lions, J.-L., 1971, Optimal control of systems governed by partial differential equations. *Grundlehren der Mathematischen Wissenschaften*, Vol. 170 (Berlin: Springer-Verlag).

[8] Fursikov, A.V., 1999, Optimal control of distributed systems: theory and applications. *Translations of Mathematical Monography*, Vol. 187 (Providence: AMS).

[9] Litvinov, W.G., 2000, Optimization in elliptic problems with applications to mechanics of deformable bodies and fluid mechanics. *Operational Theory: Advances and Applications*, Vol. 119 (Basel: Birkhäuser Verlag).

[10] Tröltzsch, F., 2005, *Optimale Steuerung partieller Differentialgleichungen* (Wiesbaden: Friedr. Vieweg & Sohn Verlag).

[11] Hinze, M. and Kunisch, K., 2001, Second order methods for optimal control of time-dependent fluid flow. *SIAM Journal on Control and Optimization*, **40**, 925–946.

[12] Tröltzsch, F., 1999, On the Lagrange-Newton-SQP method for the optimal control of semilinear parabolic equations. *SIAM Journal on Control and Optimization*, **38**, 294–312.

[13] Estep, D. and Larsson, S., 1993, The discontinuous Galerkin method for semilinear parabolic problems. *RAIRO Modelisation Mathematical Analysis and Numerical Methods*, **27**, 35–54.

[14] Ciarlet, P.G., 2002. The finite element method for elliptic problems. *Classics Applied Mathematics*, Vol. 40 (Philadelphia: SIAM).

[15] Schmich, M. and Vexler, B., 2006, Adaptivity with dynamic meshes for space-time finite element discretizations of parabolic equations. *SIAM Journal of Scientific Computing*, Submitted.

[16] Walther, A. and Griewank, A., 2004, Advantages of binomial checkpointing for memory-reduced adjoint calculations. In: M. Feistauer, V. Dolejší, P. Knobloch and K. Najzar (Eds) *Numerical Mathematics and Advanced Applications* (Berlin: Springer), pp. 834–843. Proceeding of ENUMATH 2003.

[17] Becker, R., 2001, *Adaptive Finite Elements for Optimal Control Problems*. Habilitationsschrift, Institut für Angewandte Mathematik, Universität Heidelberg.

[18] Becker, R., Meidner, D. and Vexler, B., 2005, RoDoBo: A C++ library for optimization with stationary and nonstationary PDEs with interface to Gascoigne [19]. Available online at: http://www.rodobo.uni-hd.de.

[19] Becker, R., Braack, M., Meidner, D., Richter, T., Schmich, M. and Vexler, B., 2005, The finite element toolkit Gascoigne. Available online at: http://www.gascoigne.uni-hd.de.

[20] Becker, R., Dunne, T. and Meidner, D., 2005, VisuSimple: An interactive VTK-based visualization and graphics/mpeg-generation program. Available online at: http://www.visusimple.uni-hd.de.

[21] Rannacher, R. and Vexler, B., 2005, A priori error estimates for the finite element discretization of elliptic parameter identification problems with pointwise measurements. *SIAM Journal on Control and Optimization*, **44**, 1844–1863.