Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

MARTIN WEISER & SEBASTIAN GÖTSCHEL

State Trajectory Compression for Optimal Control with Parabolic PDEs

ZIB-Report 10-05 (2010)

State Trajectory Compression for Optimal Control with Parabolic PDEs

Martin Weiser Sebastian Götschel

Abstract

In optimal control problems with nonlinear time-dependent 3D PDEs, full 4D discretizations are usually prohibitive due to the storage requirement. For this reason gradient and Newton type methods working on the reduced functional are often employed. The computation of the reduced gradient requires one solve of the state equation forward in time, and one backward solve of the adjoint equation. The state enters into the adjoint equation, again requiring the storage of a full 4D data set. We propose a lossy compression algorithm using an inexact but cheap predictor for the state data, with additional entropy coding of prediction errors. As the data is used inside a discretized, iterative algorithm, lossy compression maintaining a certain error bound turns out to be sufficient.

AMS MSC 2010: 49M29, 65K10, 65M60, 94A29

Keywords: optimal control, adjoint gradient computation, trajectory storage

1 Introduction

For the numerical solution of optimal control and variational data assimilation problems governed by nonlinear, time-dependent PDEs on a 3-dimensional spatial domain, methods working on the reduced objective functional are often employed. For the computation of the reduced gradient, one forward solve of the state equation as well as one backward solve of the adjoint equation is needed. As the state enters into the adjoint equation, the forward solution, a 4D data set, has to be stored. For large scale problems, or fine time-discretization, uncompressed storage makes this approach difficult.

But not only the sheer size of the data is a problem: as CPUs are getting rapidly faster, memory access begins to be a bottleneck for large-scale computations. Reducing the memory requirements leads to shorter idle times of the CPU while data is written to or read from mass storage devices.

So-called checkpointing techniques have been developed to overcome this problem. The state is stored only at selected timesteps, from where on parts of the trajectory are recomputed as needed. For the choice of these checkpoints, several strategies exist. In [1], an approach called multilevel windowing has been developed, based on a uniform distribution of the checkpoints. A more sophisticated strategy, binomial checkpointing, yields a minimal amount of recomputations [5], if the number of timesteps and checkpoints is fixed beforehand.

For l timesteps with varying temporal complexity, an optimal binomial checkpoint distribution with c checkpoints can be calculated in $\mathcal{O}(cl^2)$ operations. For adaptive time-stepping schemes, a heuristic has been proposed in [7], which causes only a slight increase in runtime compared to static optimal checkpointing. Recent work is concerned with optimality for a variable number of time steps ([18]). Optimality is also lost, if the number of checkpoints is not fixed, for example due to adaptive grid refinement during the forward solution.

Checkpointing techniques allow to restore the discretized state trajectory exactly. The trade-off is between storage demand, storage bandwidth, and CPU time needed for multiple forward solves. In an iterative optimization algorithm using discretized and hence inexact state trajectories, exact reconstruction is not required for convergence. Hence we propose to employ lossy compression of state trajectories, which can be reconstructed up to a sufficiently small error during the adjoint solve without incurring the need for repeated forward solves. Employing computationally inexpensive compression schemes, the trade-off is then between storage demand and accuracy.

Both lossless and lossy compression of PDE solutions mostly on structured grids have been proposed, however, without being tailored to adjoint gradient computations.

Lindstrom and Isenburg [9] suggest to traverse the simulation data in some coherent order (e. g. row-by-row on a structured grid), and perform prediction based on a subset of the already encoded data. They make no use of the geometry of the domain, or the hierarchy of grids arising from adaptive (or uniform) grid refinement in the course of the PDE solution. Shafaat and Baden [16] propose a lossy compression strategy called adaptive coarsening. Starting from a fine space discretization, their algorithm works by tentatively coarsening the mesh, reconstructing the result, and removing points where the reconstructed data approximates the original data with sufficient accuracy. If, however, the mesh is constructed by an adaptive refinement procedure during the solution progress for a given accuracy, virtually no coarsening is possible without violating the error bound. Schröder-Pander et. al. [15] make use of a sequence of tessellations to reduce storage requirement of cell averages in finite volume methods. Prediction errors are stored for each level at full accuracy, but dropping those cells with a prediction error below a prescribed tolerance.

The best-known lossy compression approach to time-varying data can be found in the MPEG video compression standard [11]. Videos consist of a series of single frames showing spatial and temporal correlations. The spatial correlations are reduced by the discrete cosine transform applied to blocks of typically 8×8 or 16×16 pixels. The resulting coefficients are then quantized in a way to maintain a certain optical quality of the video. For example, the human visual system is more sensitive to low spatial frequencies than high spatial frequencies, allowing for a coarser quantization of high frequency components. Quantization is done by dividing the coefficients by predefined factors and a rounding step. Motion prediction is performed to construct a frame from previous (and possibly later) frames, as mostly only small changes occur from one frame to the next. Typically, the encoding process, in particular the motion compensation, is rather time-consuming. There is a vast amount of literature on video compression, we refer to [11, 14, 19] and the references therein.

The approach proposed in this paper is adapted to the specific needs of adjoint gradient computation and hence differs from the algorithms above in

several ways. It is designed to be easily used on unstructured, adaptively refined grids in two and three space dimensions. We aim to control the error in the reduced gradient needed for the solution of optimal control problems. During the adjoint solve, the stored data is accessed only backwards in time, which will be exploited for temporal prediction. Compared to checkpointing, the aim is to reduce the computational overhead required for state reconstruction, such that only computationally inexpensive compression and decompression schemes will be used.

This paper is organized as follows: The problem setting and the optimization algorithm is specified in Section 2. In Section 3, the lossy compression algorithm is described in detail. Section 4 is dedicated to a discussion of the trade-offs between computation time, storage demand, and accuracy. Numerical examples are given in Section 5.

2 Problem Setting

We consider the abstract optimal control problem

$$\min_{y\in Y, u\in U} J(y,u) \text{ subject to } c(y,u)=0,$$

with the equality constraint $c : Y \times U \to Z^\star$ being a parabolic equation on Hilbert spaces Y, U, Z. Denoting with $y(u)$ the (at least locally) unique solution of the state equation for a given control u, the reduced cost functional is defined as $j(u) = J(y(u), u)$. Minimizing j allows to use any solver for the state equation as a black-box and avoids the need for a full 4D-discretization of the original optimal control problem. As we consider the unconstrained case, i. e. there are no control constraints, the first order necessary optimality conditions for a local minimizer $\bar{u} \in U$ are given as $j'(\bar{u}) = 0$.

The derivative of the reduced functional can be computed via the implicit function theorem, and is given as

$$\langle j'(u), \delta u\rangle_{U^\star,U} = \langle J_u(y(u),u) + y'(u)^\star J_y(y(u),u), \delta u\rangle_{U^\star,U}$$

with

$$y'(u) = -c_y(y(u),u)^{-1} c_u(y(u),u).$$

Defining the adjoint state λ as the solution of

$$c_y(y(u),u)^\star \lambda = J_y(y(u),u),$$

which is a backward-in-time parabolic equation, we get the derivative

$$j'(u) = J_u(y(u),u) - c_u(y(u),u)^\star \lambda.$$

Depending on the objective functional and the state equation, the solution of the forward state equation over the whole space-time cylinder is needed for the adjoint equation, incurring the need of storing the state at every timestep.

A simple class of methods for computing solutions of the optimization problem are descent methods. In combination with additional requirements on the step size, convergence can be shown if the descent directions δj satisfy the so-called angle-condition

$$\langle j'(u), \delta j\rangle_{U^\star,U} \leq -\alpha \left\|j'(u)\right\|_{U^\star} \left\|\delta j\right\|_U \tag{1}$$

for some fixed $\alpha > 0$ [6]. Defining the reduced gradient $\nabla j(u)$ with aid of the Riesz isomorphism, $\delta j = -\nabla j(u)$ is clearly an admissible descent direction. For efficient numerical implementation, discretization, numerical quadrature and iterative solution of the arising linear equation systems is necessary. Hence the reduced gradient can not be computed exactly, i. e. only $\delta j = -\nabla j(u) + e$ can be chosen. For convergence of the inexact gradient method, we have the following lemma.

Lemma 2.1. *Let $\varepsilon < \frac{1}{2}$ and compute $\delta j = -\nabla j(u) + e$ such that $\|e\| \leq \varepsilon \|\delta j\|$. Then δj satisfies the angle-condition* (1).

Proof. As $\|\delta j\| \leq \|-\nabla j(u)\| + \|e\| \leq \|-\nabla j(u)\| + \varepsilon \|\delta j\|$, we get

$$\begin{aligned}\langle j'(u), \delta j\rangle &= -\|\nabla j(u)\|^2 + \langle j'(u), e\rangle \leq -(1-\varepsilon)\|\nabla j(u)\|\,\|\delta j\| + \|\nabla j(u)\|\,\|e\| \\ &\leq -(1-2\varepsilon)\|\nabla j(u)\|\,\|\delta j\|,\end{aligned}$$

and thus (1) with $\alpha = 1 - 2\varepsilon > 0$. □

To reduce storage size and time for memory access, we allow for another source of inexactness in the adjoint equation, besides the aforementioned discretization and iteration errors. Of course the error in the reduced gradient induced by inexact storage of the state values needs to be controlled to fulfill the conditions of Lemma 2.1. For a linear state equation, the error can be computed by solving the adjoint equation with the state quantization error as the right hand side.

Example 2.2. Consider the following linear-quadratic boundary control problem.

$$\min \frac{1}{2}\|y - y_d\|^2_{L^2(\Omega\times(0,T))} + \frac{\alpha}{2}\|u\|^2_{L^2(\partial\Omega\times(0,T))}$$

subject to

$$y_t - \Delta y = 0 \text{ on } \Omega\times(0,T), \quad \partial_\nu y + y - u = 0 \text{ on } \partial\Omega\times(0,T), \quad y(\cdot,0) = 0 \text{ in } \Omega.$$

The reduced gradient is given as

$$\nabla j(u) = \alpha u + \lambda$$

with the adjoint variable λ solving

$$-\lambda_t - \Delta\lambda = y - y_d \text{ on } \Omega\times(0,T), \quad \partial_\nu\lambda + \lambda = 0 \text{ on } \partial\Omega\times(0,T), \quad \lambda(\cdot,T) = 0 \text{ in } \Omega,$$

see for example [20].

If now the exact state is perturbed by a quantization error, $\tilde{y} = y + \delta y$, the error in the reduced gradient amounts to $\delta\lambda = \tilde{\lambda} - \lambda$, with $\tilde{\lambda}$ the adjoint solution to the right hand side $\tilde{y} - y_d$. Due to the linearity, $\delta\lambda$ is the solution of

$$-\delta\lambda_t - \Delta\delta\lambda = \delta y \text{ on } \Omega\times(0,T), \;\; \partial_\nu\delta\lambda + \delta\lambda = 0 \text{ on } \partial\Omega\times(0,T), \;\; \delta\lambda(\cdot,T) = 0 \text{ in } \Omega.$$

Thus the error norm is bounded by the norm of the quantization error in the state values, e. g. for $\delta y \in L^2(\Omega\times(0,T))$, $\|\delta\lambda\|_{L^2(0,T,H^1(\Omega))} \leq c\,\|\delta y\|_{L^2(\Omega\times(0,T))}$ [20].

If the state equation is nonlinear, the coefficients of the adjoint equation may depend on the state values, and are thus not exactly known.

For the remainder of this paper, we resort to bounding a weighted L^∞-quantization error of the state. Impact on convergence rates of optimization algorithms will be thoroughly analyzed in a future paper.

3 Lossy Compression

3.1 Paradigm

For the compression algorithm, three main ingredients are used: prediction, quantization of prediction errors, and entropy coding. First, a predictor is used to construct an approximation to the finite element solution of the state equation at the current time step. As we require the predictor to be cheap in terms of computational complexity, an inexact predictor is used. Spatial correlations are exploited using prolongation in mesh hierarchies, while temporal correlations are exploited by taking values from the next time step into account. As the adjoint equation is integrated backwards in time, these values are available. Uniform quantization of the prediction error, and entropy coding of the quantized values reduces the storage requirement at the price of a loss of information.

3.2 Multilevel compression in hierarchical meshes

Notation. We consider the following discretization of the parabolic state equation. A time stepping scheme, e. g. the linearly implicit Euler method, is used on a temporal grid $0 = t_0 < \cdots < t_F = T$. At each time step, the arising elliptic sub-problems are discretized with linear finite elements on a hierarchic mesh. To fix notation, we consider a nested family $\mathcal{T}_0 \subset \cdots \subset \mathcal{T}_l$ of triangulations, constructed from an initial triangulation $\mathcal{T}_0$ of a polygonal domain $\Omega \subset \mathbb{R}^d$. To be more precise, $\mathcal{T}_j$ is generated by j levels of refinement, either uniform, or adaptively via a posteriori error estimators. We refer to j as the *level* of the triangulation $\mathcal{T}_j$, and accordingly to $\mathcal{T}_j$ as the level-j-grid. Let S_j be the space of piecewise linear finite elements over the triangulation $\mathcal{T}_j$,

$$S_j = \{y \in C^0(\bar{\Omega}) : y \text{ is a linear polynomial on each } T \in \mathcal{T}_j\}, \tag{2}$$

and $S_h := S_l$. The set of nodes on level j is denoted by $\mathcal{N}_j$, in the following we will sometimes write $k \in \mathcal{N}_j$ instead of $x_k \in \mathcal{N}_j$. With the nodal basis of S_h,

$$\Phi = \{\varphi_i : i = 0, \dots |\mathcal{N}_l| - 1\}, \quad \varphi_i(x_k) = \delta_{ik} \text{ for } x_k \in \mathcal{N}_l, \tag{3}$$

the solution of the PDE at a fixed timestep t is represented as

$$y(t, x) = \sum_{k \in \mathcal{N}_l} y_k(t)\varphi_k(x). \tag{4}$$

Prediction. The spatial predictor makes use of the grid hierarchy. Denote by $y(t)$ the coefficients at a fixed timestep of the finite element solution of the state equation computed on the finest grid, and by $y_j(t)$ the restriction of these coefficients to the level-j-grid. In the remainder of the section we leave out the dependence on t as we are concerned with stationary compression only.

Instead of storing the coefficients y uncompressed the following operations are performed:

- for each grid level $j = 0, \dots, l$:
  1. Prolongate the solution from the previous level: $\tilde{y}_j = P_j \hat{y}_{j-1}$
  2. Calculate prediction error on level j: $e_j = y_j - \tilde{y}_j$

3. Quantize the prediction error: $q_j = Q(e_j)$
4. Reconstruct the solution: $\hat{y}_j = P_j \hat{y}_{j-1} + Q^\dagger(q_j)$

In the algorithm, the reconstructed state (from the quantized coefficients) is used for prolongation instead of the exact solution y on level l. This ensures that the decompression routine can mirror this procedure, as only information available to the decompression routine is used, and thus avoids error accumulation. A similar technique is used in MPEG video compression, where the reconstructed values (i. e. inverse quantization, inverse discrete cosine transform, and in the latest standard application of in-loop deblocking filters) are used for motion prediction in the encoder as well as in the decoder ([19]).

The simplest prolongation operator P_j is linear interpolation between the grid levels, i. e. for a node $k \in \mathcal{N}_{j+1}$,

$$P_{j+1} y_j^k = \begin{cases} y_j^k, & \text{if } k \in \mathcal{N}_j \\ \frac{1}{2}(y_j^m + y_j^n), & \text{if } k \text{ is dividing the edge between nodes } m, n \in \mathcal{N}_j. \\ 0, & \text{if } j = -1. \end{cases}$$

Of course higher order prolongation can be used for prediction as well.

Quantization. The essential step of the compression algorithm is the quantization of the prediction errors, denoted above by the operator Q. The algorithm is the following:

- Let $I_j = [e_j^{\min}, e_j^{\max}]$ be the range of the prediction errors on level j
- for every node $k \in \mathcal{N}_j \setminus \mathcal{N}_{j-1}$:
  1. calculate N_j^k, the number of quantization intervals necessary for keeping a specified error bound δ^k:
  $$N_j^k \geq \frac{|I_j|}{2\delta^k}.$$
  2. divide I_j into N_j^k equally sized subintervals and compute the subinterval index i_j^k of e_j^k

For reconstruction, a coefficient with interval index i_j^k is assigned the midpoint of its subinterval, i. e.

$$Q^\dagger(i_j^k) = e_j^{\min} + \left(i_j^k + \frac{1}{2}\right) \frac{e_j^{\max} - e_j^{\min}}{N_j^k}. \tag{5}$$

Note that for each grid level, the bounds of the range I_j have to be stored on disk as well.

The quantization routine above allows to choose the number of quantization intervals for each node individually, which makes it possible to perform spatially nonuniform quantization and to bound the quantization error in weighted L^∞-norms.

For calculating the number of quantization intervals without further information as described above, we note the following obvious observation:

Lemma 3.1. *For a node k, $|y_j^k - \hat{y}_j^k| \leq \frac{\delta^k}{2}$, with δ^k the size of the quantization intervals.*

Entropy coding. The quantized coefficients are written to disk using range encoding [10]. According to their different frequency of occurrence, the coefficients are encoded with variable-sized symbols, assigning fewer bits to more frequent coefficients. As the frequency distribution has a peak at 0, this increases the compression factor. The frequency distribution can be computed and stored before encoding, or continuously updated during encoding and decoding. While storing the frequency distribution introduces a minor overhead in storage space, numerical experience shows that it may increase the performance of the range coder and thus the overall compression factor more than updating, at least for moderate problem sizes.

3.3 Backwards in time prediction

Up to now, only spatial correlations have been exploited for compression. But of course the solution at each timestep depends on the previous times, and is usually quite similar to the preceding one. As no random access is required, the temporal correlation can be used to construct a second predictor, and use differential encoding to further reduce the storage requirement. In the simplest case, the temporal predictor assumes the quantized spatial prediction error to be constant from one timestep to the next, i. e. the difference to be entropy coded is calculated for $t_n < T$ as

$$d_j^k(t_n) = \begin{cases} i_j^k(t_n) - i_j^k(t_{n+1}), & \text{if } k \in \mathcal{N}_j(t_n) \cap \mathcal{N}_j(t_{n+1}) \\ i_j^k(t_n), & \text{otherwise} \end{cases}. \tag{6}$$

Care has to be taken, as grids may change between timesteps, if adaptive refinement is used. At final time,

$$d_j^k(T) = i_j^k(T) \quad \forall k \in \mathcal{N}_j(T). \tag{7}$$

This ensures that decoding is possible backward in time. The number of different values to be encoded is reduced by this approach, increasing the performance of the range coder.

For avoiding error accumulation due to quantization, the prediction at the timestep t_n is performed not for the solution $\hat{y}(t_n)$, but for the quantized coefficients of the prediction error. Note that when using backwards in time prediction, the quantized finite element solution of at least one previous timestep has to be kept in memory, as it is encoded only after the following timestep is performed. For higher order predictors (linear, quadratic), more than two timesteps have to be kept in memory, which is easily implemented, but only feasible if the spatial discretization is not too large.

3.4 Reconstruction

With adaptive mesh refinement and time stepping, interpolation of the reconstructed state values in space and time will be needed for the adjoint solution. When storing the state values, the mesh needs to be stored additionally, see [8] for an efficient method. During the adjoint computation, at a given timestep first the corresponding state time needs to be found, $t_{\text{state}} = T - t_{\text{adjoint}}$. Since the time grids will in general be different, $y(t_{\text{state}})$ can be evaluated e. g. by constant or linear interpolation from the nearest state time steps for which the

solution was stored. Secondly, the state mesh needs to be restored, such that prediction, de-quantization and correction is performed at the correct nodes. In space, the reconstructed finite element solution of the forward equation can be evaluated by interpolation.

4 Quantization Error and Compression Rate

In this section, we will analyze the behaviour of our algorithm, and compare it to checkpointing, the current state-of-the-art method. The compression factor, or rate, measuring the performance of the algorithms is defined as the ratio between uncompressed and compressed storage size.

4.1 A priori estimates for lossy compression

As seen in Section 3.2, the trade-off between compression and quantization error depends on the range of prediction errors on each level. For ease of presentation, we look at a simple model problem on a 2D domain. As in this section we are concerned with spatial prediction only, we leave out the time dependence, assume $y \in W^{2,\infty}(\Omega)$, and use linear interpolation for prolongation. The semi-norm $|\cdot|_{2,\infty,\Omega}$ is given by

$$|y|_{2,\infty,\Omega} = \max_{|\alpha|=2} \|\partial^\alpha y\|_{L^\infty(\Omega)},$$

with a multi-index α.

Let $\Omega \subset \mathbb{R}^2$ be a polygonal domain, and $\mathcal{T}_0, \dots, \mathcal{T}_l$ a nested family of triangulations of Ω as before, with $\mathcal{T}_j$ generated from $\mathcal{T}_0$ by j *uniform* refinement steps, i. e. every triangle on level $j-1$ is subdivided into four congruent triangles in the jth refinement step. The maximum diameter of a triangle on level j is given by $h_j = \max_{T\in\mathcal{T}_j} \operatorname{diam}(T)$.

From standard finite element theory, see e. g. [3], an estimate for the interpolation error is known:

Lemma 4.1. *Let $\mathcal{T}_j$ be a shape-regular family of triangulations of a polyhedral domain Ω, and denote by $I_j := I_{h_j}$ the interpolation operator with linear polynomials. Then for $y \in W^{2,\infty}(\Omega)$,*

$$\|y - I_j y\|_{L^\infty(\Omega)} \le c h_j^2 |y|_{2,\infty,\Omega}. \tag{8}$$

For uniform refinement, $h_j = h_0 2^{-j}$ with given initial mesh-width h_0. For an a priori estimate of the error-to-compression ratio of the lossy compression algorithm, we are interested in the prediction error on level $j+1$,

$$\|I_{j+1} y - I_j y\|_{L^\infty(\Omega)},$$

as the range of the prediction error determines the number of bits needed to keep a given error bound.

Lemma 4.2. *With the same assumptions as in Lemma 4.1, it holds*

$$\|(I_j - I_{j-1})y\|_{L^\infty(\Omega)} \le 4c \frac{1}{2^{2j}} |y|_{2,\infty,\Omega}, \tag{9}$$

with c independent of j.

Proof. With a generic constant c independent of h_j,

$$\begin{aligned}\|(I_j - I_{j-1})y\|_{L^\infty(\Omega)} &\leq \|y - I_{j-1}y\|_{L^\infty(\Omega)} \leq ch_{j-1}^2|y|_{2,\infty,\Omega} \\ &\leq 4ch_j^2|y|_{2,\infty,\Omega} \leq 4c\frac{1}{2^{2j}}|y|_{2,\infty,\Omega}.\end{aligned}$$

□

Let S_h be the space of piecewise linear finite elements over the family of triangulations defined above, and consider the hierarchical basis splitting $S_h = V_0 \oplus \cdots \oplus V_l$, with $V_j = \operatorname{span}\{\psi_{jk} : k \in \mathcal{N}_j \backslash \mathcal{N}_{j-1}\}$. Here, $\mathcal{N}_{-1} = \emptyset$. With the notation introduced in Section 3 the hierarchical basis ψ_{jk} is given as follows:

$$\begin{aligned}\psi_{0k}(x_i) &= \varphi_k(x_i), \quad x_i \in \mathcal{N}_0 \\ \psi_{jk}(x_i) &= \varphi_k(x_i), \quad x_i \in \mathcal{N}_j \backslash \mathcal{N}_{j-1},\end{aligned}$$

see [21]. Hence, $y_h \in S_h$ can be written as

$$y_h = \sum_{j=0}^{l} \sum_{k \in \mathcal{N}_j \backslash \mathcal{N}_{j-1}} a_{jk}\psi_{jk}. \tag{10}$$

This decomposition yields the coefficients

$$a_{0k} = (I_0 y)(x_k), \quad x_k \in \mathcal{N}_0 \tag{11}$$

$$a_{jk} = (I_j y - I_{j-1}y)(x_k), \quad x_k \in \mathcal{N}_j \backslash \mathcal{N}_{j-1}. \tag{12}$$

With Lemma 4.2 we can estimate the ℓ^∞-norm of the coefficients of the hierarchical basis representation on a given level $j > 0$,

$$\|(a_{jk})_k\|_{\ell^\infty} \leq c2^{-2(j-1)}|y|_{2,\infty,\Omega}. \tag{13}$$

Quantization is chosen such that a given error bound δ is maintained, yielding an interval length 2δ, and thus at most

$$\frac{2\left(4ch_j^2|y|_{2,\infty,\Omega} + \delta\right)}{2\delta} = \frac{4ch_j^2|y|_{2,\infty,\Omega}}{\delta} + 1 \tag{14}$$

different quantized values on a given level j. The additive factor 2δ in the numerator on the left is due to the inexact storage of the nodes on level $j-1$, which will in the worst case differ from the original nodes by this value, see Figure 1 for a sketch of the situation. For reaching a given discretization accuracy, l refinements are needed. Allowing a quantization error of the same magnitude as the interpolation error yields

$$\delta = \|y - I_l y\|_{L^\infty(\Omega)} \leq ch_l^2|y|_{2,\infty,\Omega}. \tag{15}$$

Thus, the number of quantized values on level j can be estimated as

$$\frac{4ch_j^2|y|_{2,\infty,\Omega}}{ch_l^2|y|_{2,\infty,\Omega}} + 1 = 2^{2(l-j+1)} + 1.$$

Each value can be stored using

$$\operatorname{ld}(2^{2(l-j+1)} + 1) \leq 2(l - j + 1) + \frac{1}{2^{2(l-j+1)}}$$

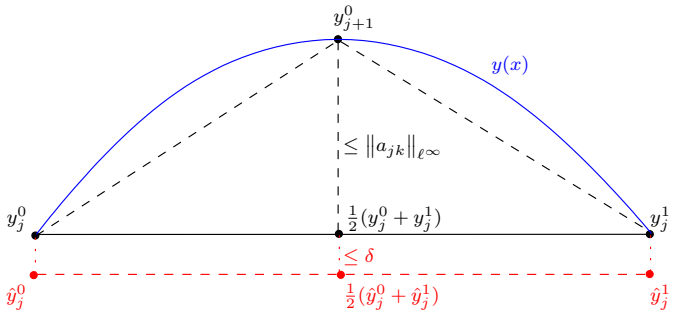

Figure 1: Prediction and quantization error

bits, where the estimate is due to the concavity of the logarithm. If higher accuracy is desired, the number of bits can be estimated by scaling δ in the previous computation.

Remark 4.3. The L^∞-approximation error for discretization by linear finite elements can be estimated as

$$\|y - y_h\|_{L^\infty(\Omega)} \leq ch^2 |\ln h| |y|_{2,\infty,\Omega}, \tag{16}$$

which behaves like $\mathcal{O}(h^{2-\varepsilon})$ for any $\varepsilon > 0$ ([3],[2]). Hence, the number of bits needed for achieving discretization error accuracy will be slightly smaller than estimated above.

For a uniformly refined grid, there are approximately $c2^{dj}$ vertices on level j, with $c2^{dj} - c2^{d(j-1)}$ new vertices on that level. The overall number of bits needed can thus be estimated as

$$\begin{aligned} &\sum_{j=1}^{l} c\left(2^{dj} - 2^{d(j-1)}\right)\left(2(l-j+1) + \frac{1}{2^{2(l-j+1)}}\right) + c\left(2(l+1) + \frac{1}{2^{2(l+1)}}\right) \\ &\quad \approx c2^{dl}\left(\frac{2^{3+3d} - 13\cdot 2^{2d} + 5\cdot 2^d + 2^{3d} - 1}{(2^d-1)^2(2^{d+2}-1)}\right) \end{aligned} \tag{17}$$

For the last estimate, terms with $-dl$ contributing to the exponent were dropped. With $c2^{dl}$ vertices in the finest mesh, the above estimate yields for $d = 2$ approximately 2.9 bits/vertex on average, which is a compression factor of 22 compared to storing double precision floating point data at 64 bit per value. In Figure 2, the resulting relation between error and compression is shown.

Remark 4.4. If we just assume $y \in H^2(\Omega)$, it holds

$$\|y - I_h y\|_{L^\infty(\Omega)} \leq ch^{2-d/2} |y|_{H^2(\Omega)}$$

as $H^2(\Omega) \subset W^{2-d/2,\infty}(\Omega)$ for $d = 2, 3$. The estimated number of bits per vertex needed to reach discretization error accuracy can then be computed as

$$\begin{aligned} &\sum_{j=1}^{l} c\left(2^{dj} - 2^{d(j-1)}\right)\left(\left(2 - \frac{d}{2}\right)(l-j+1) + \frac{1}{2^{(2-d/2)(l-j+1)}}\right) + \\ &\quad c\left(\left(2 - \frac{d}{2}\right)(l+1) + \frac{1}{2^{(2-d/2)(l+1)}}\right). \end{aligned}$$

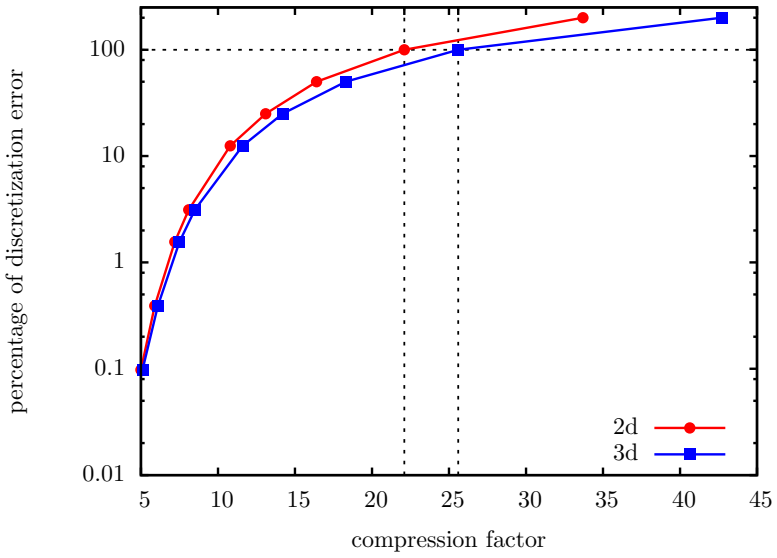

Figure 2: Error vs. compression factor: a priori estimates for spatial compression

For 2D, this yields approximately 1.8 bits/vertex. Therefore, for a fixed number of nodes, less bits are needed than in the estimate for $y \in W^{2,\infty}(\Omega)$ due to worse approximation properties.

4.2 Comparison with checkpointing

In contrast to the lossy compression approach, the trade-off of the alternative method, checkpointing, is between runtime and compression rate. For a fixed number of timesteps l to be reversed and a given number of checkpoints c, the following relation for the runtime increase is shown e. g. in [5].

Lemma 4.5. *For the minimal total number of forward steps $t(l, c)$, it holds*

$$t(l,c) = rl - \frac{(c+r)!}{(c+1)!(r-1)!},$$

with r the unique integer satisfying

$$\frac{(c+r-1)!}{c!(r-1)!} < l \leq \frac{(c+r)!}{c!r!}.$$

Thus, the ratio t/l measures the additional number of timesteps to be computed overall, relative to the number of timesteps for a single state equation solve. In Figure 3, the resulting increase in runtime for a range of compression factors is shown, for a fixed number of timesteps ($l = 100$) and varying number of checkpoints.

The number r in Lemma 4.5, the so-called repetition number, is the maximum number of times a single forward timestep is computed, and thus an upper

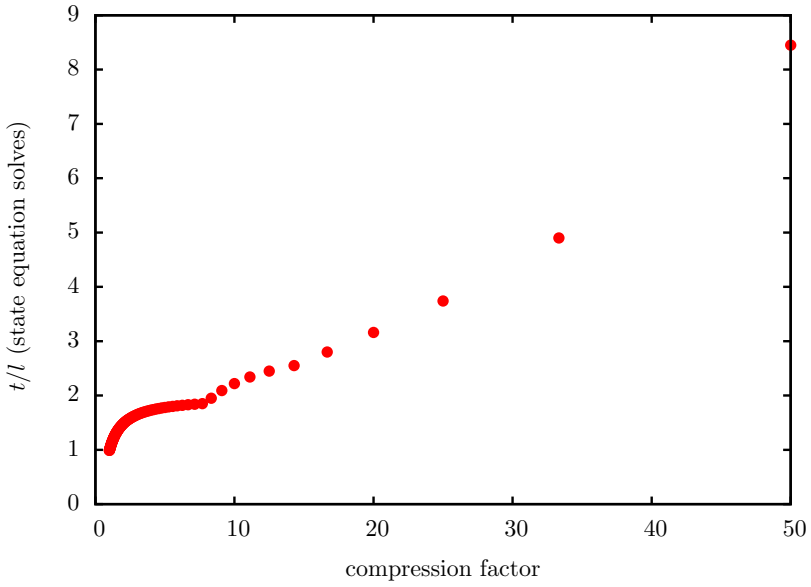

Figure 3: Relative work vs. compression factor for checkpointing, $l = 100$ timesteps

bound of the cost of checkpointing relative to the cost of a forward solve. For a given number of checkpoints, fixing r determines the maximum number of timesteps that can be reversed using binomial checkpointing.

One appealing feature of checkpointing is the slow growth of the relative work for an increasing number of timesteps. For $r \gg c = \text{const.}$, $r \approx l^{1/c}$ ([5]). This is very satisfactory for reversing a large number of timesteps, like in algorithmic differentiation, where every single arithmetic operation has to be reversed. In the context of optimal control with time-dependent PDEs, however, the number of forward timesteps is often rather small in comparison, such that the excellent limit behaviour of checkpointing is not that crucial.

The compression rate of the lossy compression approach on the other hand is more or less independent of the number of timesteps, as every frame is compressed. The only time-dependence of the compression rate enters through the temporal prediction. The smaller the timesteps, the smaller will be the temporal prediction error, and the less data will need to be stored.

Considering the size of the data which is actually written to disk, the performance of the two approaches differs clearly. The compression rates of checkpointing are reached by reusing checkpoints, such that the reduction of memory accesses is much smaller than the reduction in storage size. In [17], the write counts for checkpointing were considered, and explicit formulas derived for certain repetition rates, numbers of checkpoints and timesteps. In Figure 4, the computed write counts are shown for $l = 1000$ timesteps, and $c = 50, \ldots, 100$ checkpoints, leading to compression factors between 10 and 20. It is apparent that the amount of data transferred to the storage device, and hence the memory bandwidth, is barely reduced. However, there are settings for which

a reduction in memory bandwidth actually is achieved, e. g. for $r = 2$ and $l \leq 2c + 1$. For such a setting, the store-everything-approach, i. e. writing all timesteps of the forward solution to disk, turns out to be more expensive in terms of computation time than checkpointing, despite the need for recomputations, see [17]. Moreover, frequently accessed checkpoints can possibly be kept in RAM, decreasing the runtime. Clearly, memory bandwidth has a significant impact on the computational efficiency of the algorithm.

In contrast, the lossy compression approach reduces the memory access by the same factor as the storage space, for all compression rates.

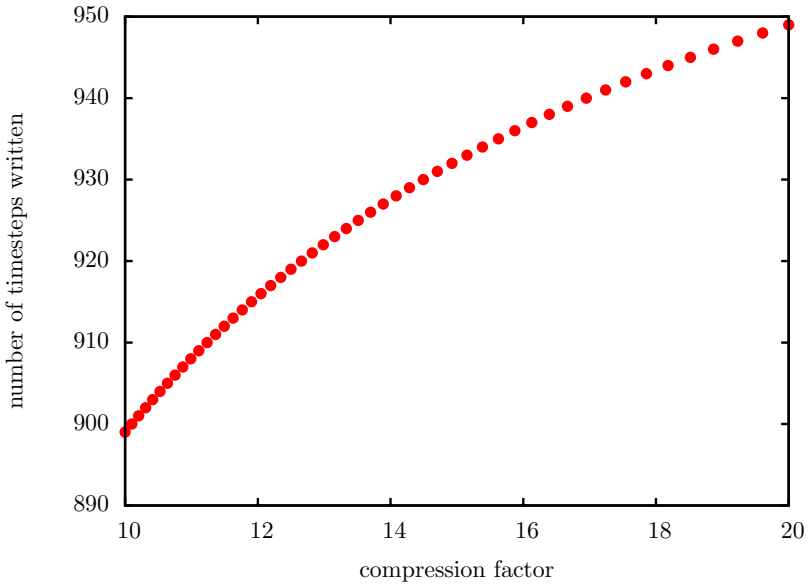

Figure 4: Actual write accesses for checkpointing and $l = 1000$ timesteps

Remark 4.6. Of course both approaches could easily be combined, using lossy compression to reduce the size of each checkpoint. As more checkpoints can be used with the same amount of available storage space, the computational work of the checkpointing algorithm is reduced. However, neither optimality of reversal schedules for dynamically varying checkpoint numbers, nor the impact of error propagation due to inexact checkpoint storage is currently clear.

4.3 Impact of adaptive mesh refinement

Using adaptive mesh refinement instead of uniformly refined grids can lead to a drastic reduction of degrees of freedom, and thus to less data. As there are less nodes on higher levels in the mesh hierarchy, the expected compression factors will be smaller than in the uniform refinement case. It might appear that adaptivity renders compression ineffective. This, however, is not true. As in the uniform refinement case, the prediction errors tend to be smaller on finer levels, such that the range to be quantized contains fewer intervals. Moreover, while adaptivity is used to control the error in the solution of the state equation,

the quantization error effects the solution of the adjoint equation. The different error propagation mechanisms might lead to different tolerances to be chosen.

A numerical example can be found in Section 5.2.

5 Numerical Examples

For simplicity of exposition, we restrict ourselves to 2D examples. The lossy compression algorithm is included in the finite element toolbox KASKADE 7 [4], which is used for the solution of the optimal control problems.

5.1 Some auxiliary test functions

To demonstrate the effectivity of the lossy compression in a simple setting, we consider the same two functions as in [15],

$$f_1(x) = \sin(12(x_0 - 0.5)(x_1 - 0.5))$$
$$f_2(x) = \begin{cases} \sin(x_0)\cos(x_1), & x_0 > 0.5 \\ \cos(x_0)\sin(x_1), & \text{otherwise} \end{cases}$$

as well as two additional functions with different curvatures,

$$f_3(x) = \sin(50(x_0 - 0.5)(x_1 - 0.5))$$
$$f_4(x) = \frac{1}{2}(x_0^2 + x_1^2)$$

As the functions only depend on $x \in [0,1]^2$, not on time, only spatial prediction and lossy encoding of the prediction errors is performed. All functions are interpolated with linear finite elements on different grids. The grids are generated by uniform red refinement from an initial coarse mesh with 2 elements, resulting in 32768 cells and 16641 nodes on the finest level for seven refinements, 131072 cells/66049 vertices for eight refinement steps, and 524288 elements/263169 nodes after nine refinements.

This setting allows to compare the a priori estimates from section 4 with the numerical results, except for f_2, which is discontinuous. The approximate L^∞-interpolation errors are shown in Table 1 together with the compression factors for a quantization error tolerance of the same magnitude. One can notice a rather good agreement with the a priori estimates. For function f_4, which has a very slight curvature, the linear interpolation used as a predictor performs expectedly good, leading to higher compression factors.

For comparison with [15], errors and compression factors for f_1 and f_2 are shown in Tables 2 and 3

As expected, for a fixed error bound δ the compression factor increases with the number of grid levels, as the prediction error gets smaller each level, and fewer bits need to be stored. In Figure 5, the reconstructed functions are shown for 7 refinement levels and $\delta = 10^{-2}$.

5.2 Adaptive mesh refinement

Adaptive mesh refinement can be seen as a means of data compression itself. To study the effect on the lossy storage approach, we consider adaptive interpolation of f_1 from Section 5.1. To reach an L^∞-interpolation error of $1.9 \cdot 10^{-4}$,

refine-ments	function	interpolation error	avg. bits/node	overall compression factor
7	f_1	$7.8 \cdot 10^{-4}$	2.56	22.8
	f_3	$1.69 \cdot 10^{-2}$	2.81	22.1
	f_4	$1.5 \cdot 10^{-5}$	1.55	42.3
8	f_1	$1.9 \cdot 10^{-4}$	2.56	24.9
	f_3	$4.2 \cdot 10^{-3}$	2.87	22.8
	f_4	$3.8 \cdot 10^{-6}$	1.49	48.3
9	f_1	$4.9 \cdot 10^{-5}$	2.56	26
	f_3	$1.1 \cdot 10^{-3}$	2.87	23.9
	f_4	$9.5 \cdot 10^{-7}$	1.49	49.1

Table 1: L^∞-interpolation errors and compression factors for the different test functions $f_i(x)$, $i = 1, 3, 4$. The average bits/node are counted after quantization, the overall compression factor contains some overhead like interval bounds, and benefits from entropy coding.

refinements	δ	compression factor	saved storage
7	$1.00 \cdot 10^{-6}$	3.0	66.7%
	$1.00 \cdot 10^{-5}$	5.2	80.8%
	$1.00 \cdot 10^{-4}$	9.7	89.7%
9	$1.00 \cdot 10^{-6}$	7.8	87.2%
	$1.00 \cdot 10^{-5}$	13.6	92.6%
	$1.00 \cdot 10^{-4}$	30.1	96.7%

Table 2: Errors and compression factors for test function f_1

eight uniform refinements were needed, leading to 66049 vertices. Using an adaptive grid, the degrees of freedom could be reduced to 42893, a compression factor of merely 1.5. Lossy storage of the nodal values led to overall compression factors of 24.9 in the uniform case, and 13.8 for the adaptive interpolation.

For f_1, adaptive interpolation had a rather small impact. We now consider interpolation by linear finite elements of the 2D Gaussian function

$$f(x) = \exp\left(-\frac{(x_0 - 0.5)^2 + (x_1 - 0.5)^2}{2\sigma^2}\right),$$

with $\sigma = 0.025$ on $\Omega = [0, 1]^2$. As $f(x)$ exhibits a highly local peak, adaptive mesh refinement leads to a drastic reduction of degrees of freedom, and thus of values which need to be stored. For reaching an L^∞-interpolation error of approximately 0.0015, a uniformly refined mesh consists of 263169 vertices, whereas the adaptive grid just needs 4237 nodes. This amounts to a compression factor of ≈ 62. The compression factor of the lossy storage approach for a quantization error bound of 0.0015 reduces from 54 on the uniform mesh to 12 for the adaptive grid. The latter still amounts to 91.7% saved storage space, the combination of adaptive mesh refinement and lossy storage saves 99.9% space and memory bandwidth.

refinements	δ	compression factor	saved storage
7	$1.00 \cdot 10^{-6}$	7.0	85.7%
	$1.00 \cdot 10^{-5}$	13.9	92.8%
	$1.00 \cdot 10^{-4}$	34.1	97.1%
9	$1.00 \cdot 10^{-6}$	22.6	95.6%
	$1.00 \cdot 10^{-5}$	83.9	98.8%
	$1.00 \cdot 10^{-4}$	213.3	99.5%

Table 3: Errors and compression factors for test function f_2

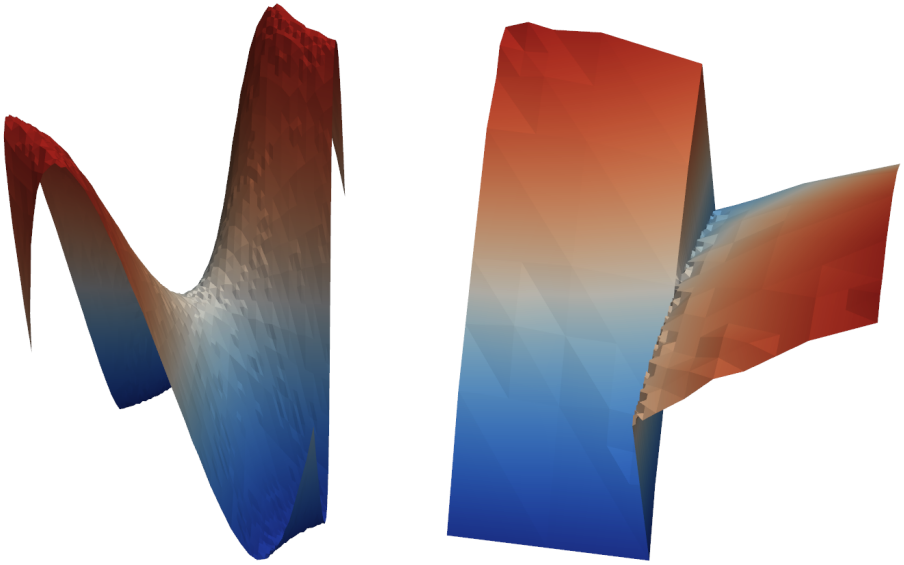

Figure 5: Reconstructed functions f_1 (left) and f_2 (right) at quantization error 10^{-2} and compression rates of 45 (f_1) and 127 (f_2) for 7 uniform refinements.

5.3 Boundary control of the linear heat equation

We now look at the model problem

$$\min \frac{1}{2} \|y - y_d\|^2_{L^2(\Omega\times(0,T))} + \frac{\alpha}{2} \|u\|^2_{L^2(\partial\Omega\times(0,T))}$$

subject to

$$\begin{aligned} y_t - \Delta y &= f \quad \text{in } \Omega \times (0,T) \\ \partial_\nu y + y &= u \quad \text{on } \partial\Omega \times (0,T) \\ y(\cdot,0) &= 0 \quad \text{in } \Omega. \end{aligned}$$

The above optimal control problem is solved with the aid of the compression algorithm in its basic form, i. e. spatial prediction by linear interpolation between the grid levels, and constant prediction in time.

The given data are

$$\Omega =]0,1[\times]0,1[, \quad T = 1, \quad \alpha = 10^{-5},$$
$$y_d(x,t) = t((x_0-1)^2 + (x_1-1)^2), \quad f(x,t) = (x_0-1)^2 + (x_1-1)^2 - 4t.$$

We apply an implicit Euler method for time stepping, with a fixed step size $dt = 0.05$, and a spatial discretization with linear finite elements on a grid with 32768 cells on the finest level, generated by 7 uniform refinement steps from the coarse grid. For minimization, a simple gradient-descent algorithm with an Armijo step size rule is used (see e. g. [6],[13]). The discretization errors in the reduced gradient and control are estimated by using a solution of the problem on a fine mesh as reference. For comparison, we stop the optimization after a certain number of iterations. We notice an increase in runtime of about 15% for a compression rate of 33.1 (i. e. 96.9% of the memory is saved) and a relative L^∞-error in the reduced gradient of $3.35 \cdot 10^{-5}$, well below the relative discretization error of approximately $3.31 \cdot 10^{-4}$ (see Figure 6). The qualitative behaviour predicted by the a priori estimates is clearly visible. Also, the impact of the simple temporal prediction can be seen. Note that despite the relatively small size all state values are written to disk, and are not kept in RAM. The runtime-increase is due to the small problem size and will of course diminish for larger problems, where the writing takes more time. Eventually, we expect a decrease in runtime compared to uncompressed storage.

Of course the algorithm works without modifications on unstructured grids. We consider the above boundary control problem on the domain shown in Figure 7. The finest mesh is generated by 6 refinement steps, resulting in 57344 cells (29121 nodes). For the same error bound, the compression factor for the unstructured grid is not as good as for the structured grid. As there is one less refinement level for the unstructured grid, this was to be expected. Nevertheless, the results, shown in Figure 8, are quite satisfactory.

5.4 Control of a semilinear parabolic equation

We now consider an optimization problem with a semilinear parabolic equation on the unit square, with the time-dependent control acting in five parts of the domain (see Figure 9). The control is only varying in time and is constant in space on each of the five shaded subdomains.

The optimal control problem is given by

$$\min \frac{1}{2} \|y - y_d\|^2_{L^2(\Omega\times(0,T))} + \frac{\alpha}{2} \|u\|^2_{L^2(0,T;\mathbb{R}^5)}$$

subject to

$$\begin{aligned} y_t - \varepsilon^2 \Delta y = y(y-a)(1-y) + u \quad & \text{in } \Omega \times (0,T) \\ \partial_\nu y = 0 \quad & \text{on } \partial\Omega \times (0,T) \\ y(\cdot,0) = y_0 \quad & \text{in } \Omega \end{aligned}$$

and

$$\Omega =]0,1[\times]0,1[, \quad T = 10, \quad a = 0.1, \quad \alpha = 10^{-5}, \quad \varepsilon = 0.15$$
$$y_d(x,t) = \frac{1}{1 + e^{\left((\|x\| - \frac{1}{3}) \cdot \frac{1}{\varepsilon\sqrt{2}} - t\right)}}, \quad y_0(x) = y_d(x,0).$$

This problem can be seen as a mock-up of optimization of cardiac defibrillation for the monodomain equation, see e. g. [12], where the control is some external current density stimulus applied only at parts of the domain.

As before, the optimal control problem is solved by a steepest descent method, with the PDEs being discretized in time by a linearly implicit, extrapolated Euler method with fixed step size $dt = 0.1$, and linear finite elements in space. The grid hierarchy consists of 8 levels, with 32768 cells and 16641 nodes on the finest level. Again, we obtain good compression rates with relative errors in the reduced gradient and computed control well below the discretization error (see Figure 10), approximated as in Example 5.3. The optimization algorithm is stopped after some iterations to be able to compare results for different compression rates. Note that here, as before, the quantization error tolerance is fixed, and not chosen according to the current size of the gradient norm in each iteration.

Conclusion

In this paper we have presented an algorithm for the lossy compression of state trajectories as needed for adjoint gradient computation. A considerable reduction of the required space can be achieved at negligible computational cost. The errors introduced in the reduced gradient are well below the discretization error for a wide range of practically relevant compression factors. Therefore, lossy compression appears to be a viable alternative to checkpointing.

Acknowledgement. Partial funding by the DFG Research Center MATHEON, project F9, is gratefully acknowledged.

References

[1] Roland Becker, Dominik Meidner, and Boris Vexler. Efficient numerical solution of parabolic optimization problems by finite element methods. *Optim. Methods Softw.*, 22(5):813–833, 2007.

[2] Philippe G. Ciarlet. *The finite element method for elliptic problems.* North-Holland, Amsterdam, 1978.

[3] Alexandre Ern and Jean-Luc Guermond. *Theory and Practice of Finite Elements.* Springer, New York, 2004.

[4] S. Götschel, M. Weiser, and A. Schiela. Solving optimal control problems with the Kaskade 7 finite element toolbox. ZIB Report 10-25, 2010.

[5] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation.* SIAM, Philadelphia, 2008.

[6] Michael Hinze, Rene Pinnau, Michael Ulbrich, and Stefan Ulbrich. *Optimization with PDE constraints.* Springer, Berlin, 2009.

[7] Michael Hinze and Julia Sternberg. A-revolve: an adaptive memory-reduced procedure for calculating adjoints; with an application to computing adjoints of the instationary navier-stokes system. *Optim. Methods Softw.*, 20(6):645–663, 2005.

[8] Felix Kälberer, Konrad Polthier, and Christoph von Tycowicz. Lossless compression of adaptive multiresolution meshes. In *Proc. Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI)*, volume 22, 2009.

[9] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, 2006.

[10] G.N.N. Martin. Range encoding: an algorithm for removing redundancy from a digitised message. Presented at Video & Data Recording Conference, Southampton, 1979.

[11] Joan L. Mitchell, William B. Pennebaker, Chad E. Fogg, and Didier J. LeGall, editors. *MPEG video compression standard*. Chapmann & Hall, New York, 1997.

[12] Chamakuri Nagaiah, Karl Kunisch, and Gernot Plank. Numerical solution for optimal control of the reaction-diffusion equations in cardiac electrophysiology. *Comput. Optim. Appl., to appear*.

[13] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, 2006.

[14] Iain E. G. Richardson. *Video codec design*. Wiley, Chichester, 2002.

[15] Friederike Schröder-Pander, Thomas Sonar, and Oliver Friedrich. Generalized multiresolution analysis on unstructured grids. *Numer. Math.*, 86(4):685–715, 2000.

[16] Tallat M. Shafaat and Scott B. Baden. A method of adaptive coarsening for compressing scientific datasets. In Bo Kågström, Erik Elmroth, Jack Dongarra, and Jerzy Wasniewski, editors, *Proc. 8th International Workshop on Applied Parallel Computing (PARA 06)*, Umeå, Sweden, 2007. Springer.

[17] Philipp Stumm and Andrea Walther. Multi-stage approaches for optimal offline checkpointing. *SIAM J. Sci. Comput.*, 31(3):1946–1967, 2009.

[18] Philipp Stumm and Andrea Walther. New algorithms for optimal online checkpointing. *SIAM J. Sci. Comput.*, 32(1):836–854, 2010.

[19] Gary J. Sullivan and Thomas Wiegand. Video compression – from concepts to the H.264/AVC standard. *Proceedings of the IEEE*, 93(1):18–31, 2005.

[20] Fredi Tröltzsch. *Optimale Steuerung partieller Differentialgleichungen*. Vieweg, Wiesbaden, 2005.

[21] Harry Yserentant. On the multi-level splitting of finite element spaces. *Numer. Math.*, 49(4):379–412, 1986.

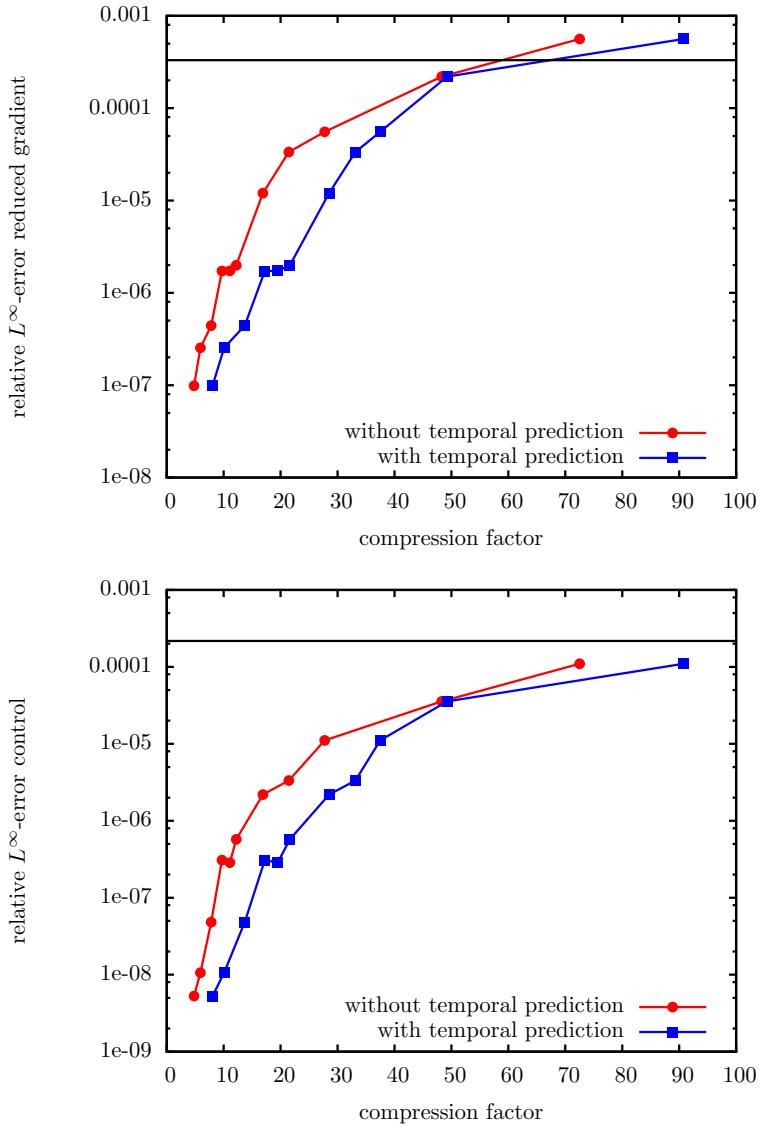

Figure 6: Relative error vs. compression rate for gradient (top) and control (bottom) after 100 iterations for the linear problem, for different tolerances for the quantization error. The horizontal line shows the approximated discretization error. The temporal predictor yields a noticeable increase of the compression factor at virtually no computational cost.

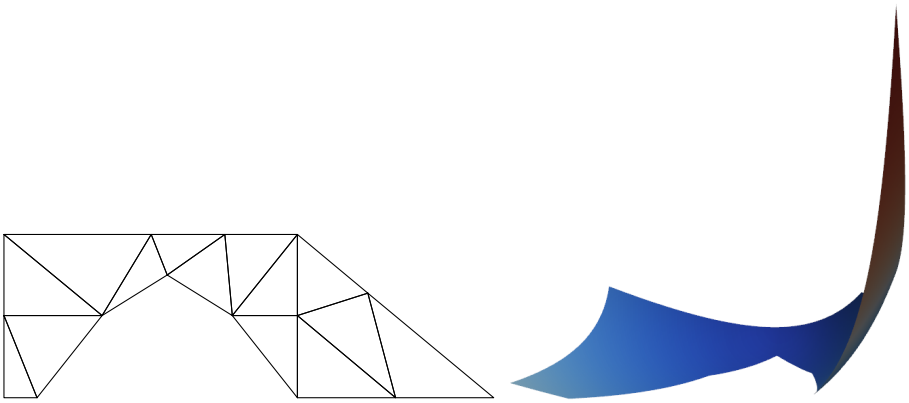

Figure 7: Initial triangulation of Ω for example 1b and desired state at final time.

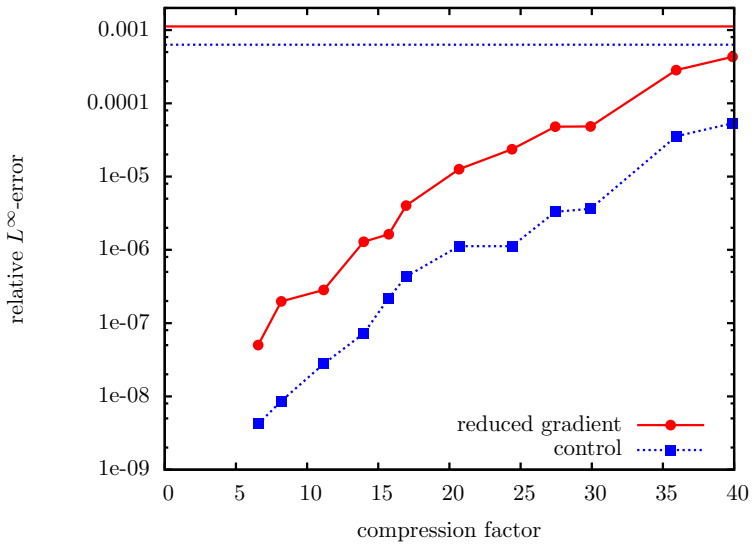

Figure 8: Relative error vs. compression rate for reduced gradient and control after 100 iterations for the linear problem on the unstructured grid (example 1b). The horizontal lines show the approximated discretization errors for the reduced gradient (solid) and control (dashed).

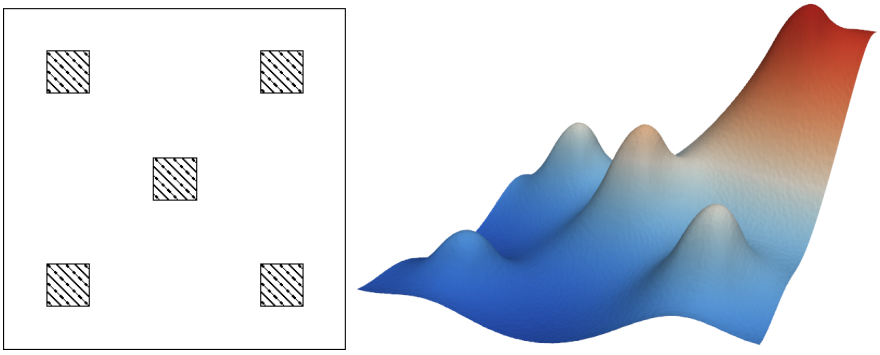

Figure 9: Domain Ω for example 2 and reconstructed state at $t = 0.8$ after 50 iterations ($\delta = 5 \cdot 10^{-4}$, compression factor 56, 1.14 bits/double).

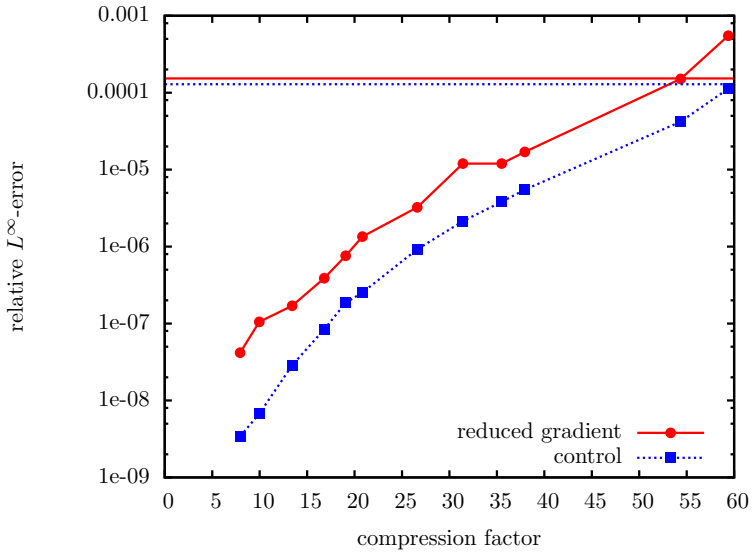

Figure 10: Relative error vs. compression rate for reduced gradient and control after 50 iterations for the semilinear problem. The horizontal line shows the approximated discretization errors for the reduced gradient (solid) and the control (dashed).