SIAM J. Optim.
Vol. 31, No. 2, pp. 1242–1275

RANDOMIZED SKETCHING ALGORITHMS FOR LOW-MEMORY DYNAMIC OPTIMIZATION[*]

RAMCHANDRAN MUTHUKUMAR[†], DREW P. KOURI[‡], AND MADELEINE UDELL[†]

Abstract. This paper develops a novel limited-memory method to solve dynamic optimization problems. The memory requirements for such problems often present a major obstacle, particularly for problems with PDE constraints such as optimal flow control, full waveform inversion, and optical tomography. In these problems, PDE constraints uniquely determine the state of a physical system for a given control; the goal is to find the value of the control that minimizes an objective. While the control is often low dimensional, the state is typically more expensive to store. This paper suggests using randomized matrix approximation to compress the state as it is generated and shows how to use the compressed state to reliably solve the original dynamic optimization problem. Concretely, the compressed state is used to compute approximate gradients and to apply the Hessian to vectors. The approximation error in these quantities is controlled by the target rank of the sketch. This approximate first- and second-order information can readily be used in any optimization algorithm. As an example, we develop a sketched trust-region method that adaptively chooses the target rank using a posteriori error information and provably converges to a stationary point of the original problem. Numerical experiments with the sketched trust-region method show promising performance on challenging problems such as the optimal control of an advection-reaction-diffusion equation and the optimal control of fluid flow past a cylinder.

Key words. PDE-constrained optimization, matrix approximation, randomized algorithm, single-pass algorithm, sketching, adaptivity, trust-region method, flow control, Navier–Stokes equations, adjoint equation

AMS subject classifications. 49M37, 49L20, 68W20, 90C30, 90C39, 93C20

DOI. 10.1137/19M1272561

1. Introduction.

In this paper, we introduce novel low-memory methods to solve discrete-time dynamic optimization problems that are based on randomized matrix sketching. Such problems arise in many practical applications, including full waveform inversion [26, 33, 38], optimal flow control [18, 28], financial engineering [22], and optical tomography [2, 23], to name a few. Let M be the dimension of the state space and m be the dimension of the control space. For many practical applications,

[*]Received by the editors July 5, 2019; accepted for publication (in revised form) January 22, 2021; published electronically May 10, 2021. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.
https://doi.org/10.1137/19M1272561

Funding: The first and third authors were supported in part by DARPA award FA8750-17-2-0101, NSF awards IIS-1943131 and CCF-1740822, the ONR Young Investigator Program, and the Simons Institute. The first and second authors were supported in part by the Laboratory Directed Research and Development program at Sandia National Laboratories.

[†]Department of Operations Research and Information Engineering, Cornell University, Ithaca, NY 14853 USA (rm949@cornell.edu, udell@cornell.edu).
[‡]Optimization and Uncertainty Quantification, Sandia National Laboratories, Albuquerque, NM 87185 USA (dpkouri@sandia.gov).

$M \gg m$. We consider the discrete-time dynamic optimization problem

$$\begin{aligned} &\underset{\mathbf{u}_n \in \mathbb{R}^M,\, \mathbf{z}_n \in \mathbb{R}^m}{\text{minimize}} && \sum_{n=1}^{N} f_n(\mathbf{u}_{n-1}, \mathbf{u}_n, \mathbf{z}_n) \\ &\text{subject to} && c_n(\mathbf{u}_{n-1}, \mathbf{u}_n, \mathbf{z}_n) = \mathbf{0}, \quad n = 1, \ldots, N, \end{aligned} \tag{1.1}$$

where $\mathbf{z}_n \in \mathbb{R}^m$, $\mathbf{u}_n \in \mathbb{R}^M$ are the control actions and system states at the nth time step, respectively, $\mathbf{u}_0 \in \mathbb{R}^M$ is the provided initial state of the system, $f_n : \mathbb{R}^M \times \mathbb{R}^M \times \mathbb{R}^m \to \mathbb{R}$ is a "cost" or "objective" associated with the nth state and control, and $c_n : \mathbb{R}^M \times \mathbb{R}^M \times \mathbb{R}^m \to \mathbb{R}^M$ is a constraint function that advances the state from $\mathbf{u}_{n-1}$ into $\mathbf{u}_n$. One major application of problem (1.1) is to optimize (a discretized version of) a continuous-time dynamical system. In this case, the form of c_n presented above corresponds to single-step time integration schemes. Other time stepping methods can also be handled with the approach described here. Additionally, our approach can handle dynamic optimization problems with static controls, including, e.g., initial conditions, material parameters, and shape or topological designs. However, for simplicity we focus on problems of the form (1.1).

1.1. Memory versus computation: Trade-offs. Memory limits often constrain numerical algorithms for (1.1). For example, suppose the objective and constraints are twice differentiable. To solve (1.1) using a traditional sequential quadratic programming algorithm, we must store the entire state trajectory $\{\mathbf{u}_n\}$, the Lagrange multipliers associated with each constraint function in (1.1), and the control trajectory $\{\mathbf{z}_n\}$: in total, we have a memory requirement of $N(2M + m)$ floating point numbers. For example, discretizations of full waveform inversion problems for petroleum exploration regularly result in state vectors of size $M = 64$ billion with the number of time steps exceeding $N = 400,000$ [27]. In view of the onerous memory requirements of straightforward algorithms, algorithm designers must make hard choices to reduce the fidelity of the model or to repeat computation.

One can reduce the storage and computational complexity, at the cost of accuracy, using coarse spatial and temporal grids to model the problem. A more ambitious approach than coarsening is to solve (1.1) using a reduced-order model (ROM) [1, 10, 21]. However, ROMs are often tailored for specific dynamical systems and demand significant domain expertise. Moreover, ROMs can be difficult to implement in practice, requiring significant and often invasive modification of the simulation software. Naively implemented, ROMs are also a poor fit for optimization. For example, proper orthogonal decomposition ROMs are constructed using snapshots of the state trajectory $\{\mathbf{u}_n\}$, which depend on the current control trajectory $\{\mathbf{z}_n\}$. Therefore, as the control changes during optimization, the approximation quality of the ROM degrades. Adaptive ROM generation for optimization is an active research topic [11, 41]. Balanced truncation is better suited for optimization since it does not depend on snapshots of the state trajectory. However, balanced truncation is typically limited to linear, time-invariant dynamical systems. See [5] for a review of ROM techniques for PDE-constrained optimization.

An alternative approach substitutes computation for memory. Suppose the dynamic constraint in (1.1) uniquely determines the state given the control and forms the equivalent reduced optimization problem by eliminating the state "nuisance variable." The optimization variable in this approach is simply the control $\{\mathbf{z}_n\}$: Nm floating point numbers. However, evaluating the objective function requires solving the dynamic constraint. Worse, evaluating the gradient of the objective function requires the solution of the backward-in-time adjoint equation [20]: to solve it, we must traverse the state trajectory backward, from the end to the beginning. Unfortunately, the state must generally be computed forward in time.

Checkpointing methods perform this backward pass without storing the full state [3, 15, 30, 37]. Instead, they store judiciously chosen snapshots of the state variables $\mathbf{u}_n$ in memory or to hard disk. The state is then recomputed from these checkpoints to solve the adjoint equation. This procedure results in lower memory requirements but drastically increases the cost of computing gradient information. For example, if we can store at most k state vectors in memory (i.e., kM floating point numbers) and we solve the dynamic optimization problem (1.1) using the checkpointing strategy described in [15] with k checkpoints, then Proposition 1 of [15] guarantees that the minimum number of additional state time steps required to perform the backward pass of the adjoint equation is

$$w(N,k) := \tau N - \beta(k+1,\tau-1), \qquad \text{where} \qquad \beta(s,t) := \binom{s+t}{s}$$

and τ is the unique integer satisfying $\beta(k,\tau-1) < N \leq \beta(k,\tau)$. This cost is compounded when higher-order derivatives are required.

Our approach is closely related to the compression approaches described in [9, 13, 14]. The method in [13, 14] applies to (1.1) with dynamic constraints given by discretized partial differential equations (PDEs). Their approach uses a sequence of nested meshes to adaptively reduce to the state storage requirement. Consequently, this method is not directly applicable to general problems with the form (1.1). On the other hand, the authors of [9] explore the use of three methods based on principal component analysis (PCA), sequential Gram–Schmidt orthogonalization (GS), and the discrete Fourier transformation (DFT) for adjoint-based error estimation of PDE systems. The PCA and GS approaches compress in space, but only provide marginal compression in time, while the DFT provides temporal compression. The authors recommend the combination of these spatial and temporal compression approaches and show compelling results for the combination of GS and DFT. In the context of PDE-constrained optimization, our approach provides both temporal and spatial compression using randomized sketching.

1.2. Randomized sketching for dynamic optimization. In contrast to the checkpointing methods, our sketching methods can achieve $\mathcal{O}(N)$ computation with $\mathcal{O}(N+M)$ storage, where the constant hidden by the big-O notation depends on the rank of the state matrix. Indeed, our methods solve the state equation only once at each iterate. The sketching method is simple and easy to integrate into existing codes: (1) compute the sketch while solving the state equation by forming a random projection, (2) reconstruct the approximate state via simple linear algebra, and (3) use the low-rank approximation in place of the state throughout the remainder of the computation, for example, to solve the adjoint equation and compute an approximate gradient. Under standard assumptions, we can quantify the effect of these approximate gradients on the quality of the approximate solution to the dynamic optimization problem (1.1). We also develop a trust-region algorithm to solve (1.1) that ensures convergence by adaptively choosing the rank.

1.3. Outline. We first introduce notation and describe the problem formulation. We then introduce a sketching method for matrix approximation and analyze the error committed when solving (1.1) with a fixed-rank sketch. Subsequently, we introduce an adaptive-rank trust-region algorithm and discuss its convergence. We verify our assumptions for a class of optimal control problems constrained by linear parabolic

PDEs. We provide numerical results for this class of problems as well as for a class of flow control problems for which the assumptions have not been verified.

2. Problem formulation.

To begin, we introduce notation for the dynamic optimization problem. We consider the control vectors $\mathbf{z}_n$ and the state vectors $\mathbf{u}_n$ to be column vectors and collect the control and state trajectories into the stacked column vectors

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_N \end{bmatrix}, \quad \mathbf{z}_n \in \mathbb{R}^m \quad \forall\, n = 1, \dots, N,$$

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_N \end{bmatrix}, \quad \mathbf{u}_n \in \mathbb{R}^M \quad \forall\, n = 1, \dots, N.$$

We denote the control and state spaces, respectively, by $\mathfrak{Z} := \mathbb{R}^{mN}$ and $\mathfrak{U} := \mathbb{R}^{MN}$. Moreover, we consider the family of coordinate projections $p_n : \mathbb{R}^{MN} \times \mathbb{R}^{mN} \to \mathbb{R}^M \times \mathbb{R}^M \times \mathbb{R}^m$ defined by

$$p_1(\mathbf{U}, \mathbf{Z}) := (\mathbf{u}_0, \mathbf{u}_1, \mathbf{z}_1) \qquad \text{and} \qquad p_n(\mathbf{U}, \mathbf{Z}) := (\mathbf{u}_{n-1}, \mathbf{u}_n, \mathbf{z}_n), \quad n = 2, \dots, N,$$

where the initial state $\mathbf{u}_0$ is given. Other choices of the projection mappings $\{p_n\}$ result in different orderings of the trajectory. These model, e.g., delays in the dynamics or different time stepping schemes. Throughout the paper, $\|\cdot\|_2$ refers to the Euclidean vector norm and $\|\cdot\|_{\mathrm{F}}$ the Frobenius matrix norm. For later results, we will require the weighted norms $\|\mathbf{v}\|_{\mathbf{A}}^2 = \mathbf{v}^\top \mathbf{A} \mathbf{v}$ for $\mathbf{v} \in \mathbb{R}^\ell$, where $\mathbf{A} \in \mathbb{R}^{\ell \times \ell}$ is a symmetric positive definite matrix. In addition, we denote the singular values of a matrix $\mathbf{B} \in \mathbb{R}^{M \times N}$ by $\sigma_{\min}(\mathbf{B}) = \sigma_1(\mathbf{B}) \le \cdots \le \sigma_{\min(M,N)}(\mathbf{B}) = \sigma_{\max}(\mathbf{B})$.

Using this notation, we can represent the dynamic constraint and objective as the functions

$$c(\mathbf{U}, \mathbf{Z}) := \begin{bmatrix} c_1 \circ p_1 \\ \vdots \\ c_N \circ p_N \end{bmatrix} (\mathbf{U}, \mathbf{Z}) \qquad \text{and} \qquad f(\mathbf{U}, \mathbf{Z}) := \sum_{n=1}^{N} f_n \circ p_n(\mathbf{U}, \mathbf{Z}),$$

where $c : \mathfrak{U} \times \mathfrak{Z} \to \mathfrak{U}$ and $f : \mathfrak{U} \times \mathfrak{Z} \to \mathbb{R}$ and we can rewrite the dynamic optimization problem (1.1) as

$$\begin{aligned} &\underset{\mathbf{U} \in \mathfrak{U},\, \mathbf{Z} \in \mathfrak{Z}}{\text{minimize}} && f(\mathbf{U}, \mathbf{Z}) \\ &\text{subject to} && c(\mathbf{U}, \mathbf{Z}) = \mathbf{0}. \end{aligned} \tag{2.1}$$

2.1. Assumptions and the reduced problem.

Throughout this paper, we will assume that f and c are continuously differentiable on $\mathfrak{U} \times \mathfrak{Z}$. In general, we denote by d_i the partial derivative of a function with respect to its ith argument. We assume that there exists a control-to-state map $S : \mathfrak{Z} \to \mathfrak{U}$ such that for any control $\mathbf{Z} \in \mathfrak{Z}$, $\bar{\mathbf{U}} := S(\mathbf{Z})$ is the unique state trajectory that satisfies the dynamic constraint,

$$c(\bar{\mathbf{U}}, \mathbf{Z}) = 0,$$

and that the state Jacobian of the constraint, $\mathsf{d}_1 c(\bar{\mathbf{U}}, \mathbf{Z})$, has a bounded inverse for all controls $\mathbf{Z} \in \mathfrak{Z}$. Note that the unique state trajectory $\bar{\mathbf{U}} = S(\mathbf{Z})$ has the form

$$S(\mathbf{Z}) := \begin{bmatrix} S_1(\mathbf{u}_0, \mathbf{z}_1) \\ S_2(S_1(\mathbf{u}_0, \mathbf{z}_1), \mathbf{z}_2) \\ \vdots \\ S_N(S_{N-1}(\dots, \mathbf{z}_{N-1}), \mathbf{z}_N) \end{bmatrix},$$

where $\bar{\mathbf{u}}_n = S_n(\bar{\mathbf{u}}_{n-1}, \mathbf{z}_n) \in \mathbb{R}^M$ denotes the unique solution to

$$c_n(\bar{\mathbf{u}}_{n-1}, \bar{\mathbf{u}}_n, \mathbf{z}_n) = 0 \quad \forall\, n = 1, \dots, N.$$

Under these assumptions, the implicit function theorem (cf. [20, Thm. 1.41]) ensures that the operators S_n and S are continuously differentiable. In addition, if c has continuous ℓth-order derivatives for $\ell \in \mathbb{N}$, then S_n and S are ℓth-order continuously differentiable. Using the control-to-state map S, we can reformulate (2.1) as the reduced dynamic optimization problem

$$\underset{\mathbf{Z} \in \mathfrak{Z}}{\text{minimize}} \quad \{F(\mathbf{Z}) := f(S(\mathbf{Z}), \mathbf{Z})\}. \tag{2.2}$$

Our goal is to solve the reduced dynamic optimization problem (2.2) efficiently. This reduced formulation is helpful when the problem size, and therefore the memory required to store the state, is large.

2.2. Gradient computation and adjoints.

We consider derivative-based optimization approaches to solve the dynamic optimization problem (2.2). These require computing first-order and (if possible) second-order derivative information. To compute the gradient of the reduced objective function F, we employ the adjoint method [20], which results from an application of the chain rule to the implicitly defined reduced objective function F. In particular, the variation of F in the direction $\mathbf{V} \in \mathfrak{Z}$ is given by

$$\begin{aligned} \langle \nabla F(\mathbf{Z}), \mathbf{V} \rangle_{\mathfrak{Z}} &= \langle \mathsf{d}_1 f(S(\mathbf{Z}), \mathbf{Z}), S'(\mathbf{Z})\mathbf{V} \rangle_{\mathfrak{U}} + \langle \mathsf{d}_2 f(S(\mathbf{Z}), \mathbf{Z}), \mathbf{V} \rangle_{\mathfrak{Z}} \\ &= \langle S'(\mathbf{Z})^* \mathsf{d}_1 f(S(\mathbf{Z}), \mathbf{Z}) + \mathsf{d}_2 f(S(\mathbf{Z}), \mathbf{Z}), \mathbf{V} \rangle_{\mathfrak{Z}}, \end{aligned}$$

where $S'(\mathbf{Z})$ denotes the derivative of the control-to-state map S at $\mathbf{Z}$ and $S'(\mathbf{Z})^*$ its adjoint. Here, $\langle \cdot, \cdot \rangle_{\mathfrak{Z}}$ and $\langle \cdot, \cdot \rangle_{\mathfrak{U}}$ denote inner products on $\mathfrak{Z}$ and $\mathfrak{U}$, respectively. The implicit function theorem ensures that $S'(\mathbf{Z})\mathbf{V}$ satisfies the linear system of equations

$$\mathsf{d}_1 c(S(\mathbf{Z}), \mathbf{Z}) S'(\mathbf{Z})\mathbf{V} + \mathsf{d}_2 c(S(\mathbf{Z}), \mathbf{Z})\mathbf{V} = 0. \tag{2.3}$$

By the assumption that the state Jacobian of the constraint, $\mathsf{d}_1 c(S(\mathbf{Z}), \mathbf{Z})$, has a bounded inverse for all control $\mathbf{Z} \in \mathfrak{Z}$, we have that (2.3) has a unique solution given by

$$S'(\mathbf{Z}) = -(\mathsf{d}_1 c(S(\mathbf{Z}), \mathbf{Z}))^{-1} \mathsf{d}_2 c(S(\mathbf{Z}), \mathbf{Z}).$$

Therefore, the adjoint of the derivative of the control-to-state map is given by

$$S'(\mathbf{Z})^* = -(\mathsf{d}_2 c(S(\mathbf{Z}), \mathbf{Z}))^* (\mathsf{d}_1 c(S(\mathbf{Z}), \mathbf{Z}))^{-*}.$$

Substituting this expression into (2.2) yields the gradient

$$\nabla F(\mathbf{Z}) = (\mathsf{d}_2 c(S(\mathbf{Z}), \mathbf{Z}))^* \bar{\mathbf{\Lambda}} + \mathsf{d}_2 f(S(\mathbf{Z}), \mathbf{Z}),$$

where *the adjoint*, $\bar{\mathbf{\Lambda}} = \Lambda(\mathbf{Z}) \in \mathfrak{U}$, is the unique trajectory that solves the *adjoint equation*:

$$(\mathsf{d}_1 c(S(\mathbf{Z}), \mathbf{Z}))^* \bar{\mathbf{\Lambda}} = -\mathsf{d}_1 f(S(\mathbf{Z}), \mathbf{Z}). \tag{2.4}$$

This discussion gives rise to Algorithm 2.1 for computing gradients of the reduced objective function F.

Algorithm 2.1 Compute gradient using adjoints.

```
Input: Control $\mathbf{Z}$
Output: Gradient of reduced objective function $\nabla F(\mathbf{Z})$
1: function GRADIENT($\mathbf{Z}$)
2:     Solve the state equation, $c(\mathbf{U}, \mathbf{Z}) = \mathbf{0}$, and denote the solution $\bar{\mathbf{U}}$
3:     Solve the adjoint equation, $(\mathsf{d}_1 c(\bar{\mathbf{U}}, \mathbf{Z}))^* \mathbf{\Lambda} = -\mathsf{d}_1 f(\bar{\mathbf{U}}, \mathbf{Z})$, and denote the
       solution $\bar{\mathbf{\Lambda}}$
4:     Compute the gradient as $\nabla F(\mathbf{Z}) = \mathsf{d}_2 f(\bar{\mathbf{U}}, \mathbf{Z}) + (\mathsf{d}_2 c(\bar{\mathbf{U}}, \mathbf{Z}))^* \bar{\mathbf{\Lambda}}$
5:     return $\nabla F(\mathbf{Z})$
```

Algorithm 2.1 hides the dynamic nature of the state and adjoint computations. In fact, we compute $\bar{\mathbf{U}}$ forward in time starting from $\mathbf{u}_1$ to $\mathbf{u}_N$. In contrast, the adjoint equation is computed backward in time. To see this, express the adjoint equation in terms of the N components f_n and c_n:

$$\begin{aligned}
\mathbf{0} &= \mathsf{d}_1 f(S(\mathbf{Z}), \mathbf{Z}) + (\mathsf{d}_1 c(S(\mathbf{Z}), \mathbf{Z}))^* \bar{\mathbf{\Lambda}} \\
&= \sum_{n=1}^{N} \mathsf{d}_1 (f_n \circ p_n)(S(\mathbf{Z}), \mathbf{Z}) + (\mathsf{d}_1 (c_n \circ p_n)(S(\mathbf{Z}), \mathbf{Z}))^* \bar{\boldsymbol{\lambda}}_n.
\end{aligned}$$

We can calculate partial derivatives of $c_n \circ p_n$ and $f_n \circ p_n$ using the chain rule. The adjoint equation then reduces to the following system of equations for $n = 1, \ldots, N$:

$$\begin{aligned}
(\mathsf{d}_2 c_N(\bar{\mathbf{u}}_{N-1}, \bar{\mathbf{u}}_N, \mathbf{z}_N))^* \boldsymbol{\lambda}_N &= -\,\mathsf{d}_2 f_N(\bar{\mathbf{u}}_{N-1}, \bar{\mathbf{u}}_N, \mathbf{z}_N), \\
(\mathsf{d}_2 c_n(\bar{\mathbf{u}}_{n-1}, \bar{\mathbf{u}}_n, \mathbf{z}_n))^* \boldsymbol{\lambda}_n &= -\,\mathsf{d}_2 f_n(\bar{\mathbf{u}}_{n-1}, \bar{\mathbf{u}}_n, \mathbf{z}_n) - \mathsf{d}_1 f_{n+1}(\bar{\mathbf{u}}_n, \bar{\mathbf{u}}_{n+1}, \mathbf{z}_{n+1}) \\
&\quad - \mathsf{d}_1 c_{n+1}(\bar{\mathbf{u}}_n, \bar{\mathbf{u}}_{n+1}, \mathbf{z}_{n+1})^* \boldsymbol{\lambda}_{n+1},
\end{aligned}$$

where $\bar{\mathbf{u}}_n = S_n(\bar{\mathbf{u}}_{n-1}, \mathbf{z}_n)$ for $n = 1, \ldots, N$. Here, information required for the solve flows backward in time from $\bar{\boldsymbol{\lambda}}_N$ to $\bar{\boldsymbol{\lambda}}_1$: in general, computing $\bar{\boldsymbol{\lambda}}_n$ requires the state vectors $\bar{\mathbf{u}}_{n-1}$, $\bar{\mathbf{u}}_n$, and $\bar{\mathbf{u}}_{n+1}$. The most straightforward computational approach is to solve the state equation and store the full state trajectory before computing the adjoint. The adjoint vectors $\bar{\boldsymbol{\lambda}}_n$ are used to form the gradient vector g_n at the nth time step as

$$g_n = \mathsf{d}_3 f_n(\mathbf{u}_{n-1}, \mathbf{u}_n, \mathbf{z}_n) + (\mathsf{d}_3 c_n(\mathbf{u}_{n-1}, \mathbf{u}_n, \mathbf{z}_n))^* \boldsymbol{\lambda}_n.$$

Both the state $\bar{\mathbf{U}}$ and adjoint $\bar{\mathbf{\Lambda}}$ are intermediate variables used to compute the gradient $\nabla F(\mathbf{Z})$, and both require MN storage. The control $\mathbf{Z}$ requires only mN storage, which is often much smaller in practical applications, i.e., $M \gg m$.

3. Low-memory matrix approximation.

Our method forms a low-memory approximation to the state matrix in order to solve the dynamic optimization problem without storing or recomputing the state matrix. In this section, we describe this approximation in detail. Given a fixed storage budget, in a single pass column-by-column over the matrix, the method collects information about the matrix from which the matrix can be accurately reconstructed. This information is called a *sketch* of the matrix. The approach we adopt in this paper forms the sketch as a random projection of the matrix. This approach has been studied extensively in the numerical analysis and theoretical computer science communities, and many variants are available [6, 7, 17, 29, 34, 35, 36, 39, 40]. In particular, the authors of [36] provide evidence that data generated from scientific simulations (including velocity fields in fluid-flow models, such as those presented in our numerics below) are efficiently compressed by sketching methods. When the state space itself has tensor product structure, a tensor sketch that respects that structure can further reduce memory requirements [31]. For concreteness (and for use in our numerical experiments), we describe the method developed in [36].

Consider a matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$ and a target rank parameter r. The method of [36] produces a low-rank matrix approximation $\hat{\mathbf{A}}$ that is (in expectation) not much farther from $\mathbf{A}$ than the best rank-r approximation, using $\mathcal{O}(r(M+N))$ storage.

Define the sketch parameters $r \le k \le s$. The quality of the approximation, and also the storage required for the sketch, increases with these parameters. In this paper, we choose $k := 2r+1$ and $s := 2k+1$, and adjust the target rank parameter r to obtain satisfactory performance. To define the sketch, fix four random linear dimension reduction maps (DRMs) with i.i.d. standard normal entries:

$$\begin{aligned} &\mathbf{\Upsilon} \in \mathbb{R}^{k\times M} \quad \text{and} \quad \mathbf{\Omega} \in \mathbb{R}^{k\times N};\\ &\mathbf{\Phi} \in \mathbb{R}^{s\times M} \quad \text{and} \quad \mathbf{\Psi} \in \mathbb{R}^{s\times N}. \end{aligned} \tag{3.1}$$

Note that other random ensembles work similarly; see [36]. The sketch of the target matrix $\mathbf{A}$ consists of

$$\begin{aligned} \mathbf{X} &:= \mathbf{\Upsilon}\mathbf{A} \in \mathbb{R}^{k\times N}, && \text{the co-range sketch;}\\ \mathbf{Y} &:= \mathbf{A}\mathbf{\Omega}^* \in \mathbb{R}^{M\times k}, && \text{the range sketch;}\\ \mathbf{Z} &:= \mathbf{\Phi}\mathbf{A}\mathbf{\Psi}^* \in \mathbb{R}^{s\times s}, && \text{the core sketch.} \end{aligned}$$

Roughly speaking, the range sketch $\mathbf{Y}$ captures the row space (top left singular vectors) of $\mathbf{A}$; the co-range sketch $\mathbf{X}$ captures the column space (top right singular vectors); and the core sketch $\mathbf{Z}$ captures their interactions (singular values). Linearity of the sketch allows us to compute it without storing the full matrix $\mathbf{A}$. Suppose $\mathbf{A} = [\mathbf{a}_1 \cdots \mathbf{a}_N]$ is presented column by column. Then we can compute the co-range sketch $\mathbf{X} = \mathbf{X}^{(N)}$ by the following recursion (with the initial estimate of the sketch set to $\mathbf{0}$):

$$\mathbf{X}^{(0)} = \mathbf{0}, \qquad \mathbf{X}^{(i)} = \mathbf{X}^{(i-1)} + \mathbf{\Upsilon}\mathbf{a}_i\mathbf{e}_i^\top, \quad i = 1,\dots,N, \tag{3.2}$$

where $\mathbf{e}_i$ is the ith unit vector, and similarly for the range sketch $\mathbf{Y}$ and core sketch $\mathbf{Z}$.

Sketch object. We use $\{\mathbf{A}\}_r$ to denote an object of the sketch class, which contains the sketch parameters k, s, the dimension reduction maps $\mathbf{\Upsilon}, \mathbf{\Omega}, \mathbf{\Phi}, \mathbf{\Psi}$, and the range, co-range, and core sketches $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$.

Storage. The sketch matrices $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ can be stored using $k(M+N)+s^2$ floating point numbers. Hence the memory required to store a sketch object with target rank parameter r is $\mathcal{O}(r(M+N)+r^2)$. When storage is limited, the DRMs can be regenerated on the fly from a random seed or generated from a random ensemble with lower storage requirements [31, 32], so we omit the DRMs from our storage calculation.

3.1. Reconstruction. To reconstruct a low-rank approximation from the sketch, compute the QR factorizations [12, Chap. 5.2] of $\mathbf{X}^*$ and $\mathbf{Y}$,

$$\begin{aligned} \mathbf{X}^* &=: \mathbf{P}\mathbf{R}_1, \quad \text{where} \quad \mathbf{P} \in \mathbb{R}^{N\times k}; \\ \mathbf{Y} &=: \mathbf{Q}\mathbf{R}_2, \quad \text{where} \quad \mathbf{Q} \in \mathbb{R}^{M\times k}. \end{aligned} \tag{3.3}$$

Use the core sketch $\mathbf{Z}$ to compute a core approximation by solving two small least-squares problems

$$\mathbf{C} := (\mathbf{\Phi}\mathbf{Q})^{\dagger}\mathbf{Z}((\mathbf{\Psi}\mathbf{P})^{\dagger})^* \in \mathbb{R}^{k\times k}. \tag{3.4}$$

Then compute a rank-k approximation of the target matrix $\mathbf{A}$ as

$$\{\{\mathbf{A}\}\}_r := \mathbf{QCP}^*. \tag{3.5}$$

This approximation can be truncated to rank r by replacing $\mathbf{C} \in \mathbb{R}^{k\times k}$ with its best rank-r approximation. For use in the dynamic optimization problem (1.1), after reconstruction we store the low-rank factors, $\mathbf{Q} \in \mathbb{R}^{M\times k}$ and $\mathbf{W} = \mathbf{CP}^* \in \mathbb{R}^{k\times N}$, in the sketch object $\{\mathbf{A}\}_r$. To reduce storage further, one can overwrite $\mathbf{X}$ and $\mathbf{Y}$ with $\mathbf{Q}$ and $\mathbf{W}$. From these, we can reconstruct the jth column (the state at the jth time step) as needed, via $\{\{\mathbf{A}\}\}_r[:, j] = \mathbf{QW}[:, j]$. Each of these operations uses storage proportional to $k(M + N)$, so the total storage complexity to approximate $\mathbf{A} \in \mathbb{R}^{M\times N}$ (in factored form) is $\mathcal{O}(k(M + N))$.

Remark 3.1. We summarize the sketch class and its methods in Algorithm A.1 in Appendix A. Briefly, the function `Sketch` acts as a class constructor (in object-oriented paradigm). The function `Initialize!` initializes the fields of the sketch object by fixing four random linear DRMs and setting the initial sketch matrices $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ to zero. When the sketch object has already been instantiated (such as in line 11, Algorithm 4.4 and line 8, Algorithm 4.5), the function `Initialize!` resets the fields according to the new rank parameter r. Thus previously allocated memory can be reused.

3.2. Intuition. We present an intuitive view of this sketching approximation method, following [17, 35, 36]. Consider a target matrix $\mathbf{A} \in \mathbb{R}^{M\times N}$. The rows of the DRM $\mathbf{\Omega}$ are the i.i.d. random vectors $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_k \in \mathbb{R}^N$. As before, define the range sketch $\mathbf{Y} = [\mathbf{y}_1 \cdots \mathbf{y}_k] \in \mathbb{R}^{M\times k}$, where each column $\mathbf{y}_i = \mathbf{A}\boldsymbol{\omega}_i \in \mathbb{R}^M$.

The columns of $\mathbf{Y}$ capture the range of $\mathbf{A}$, and with probability one, $\boldsymbol{\omega}_i \notin$ **nullspace**$(\mathbf{A})$ for any i. Hence each $\mathbf{y}_i$ can be viewed as an independent random sample from range$(\mathbf{A})$, and range$(\mathbf{Y}) \subseteq$ range$(\mathbf{A})$ with equality when Rank$(\mathbf{A}) = k$. Similarly, the co-range sketch captures the range of $\mathbf{A}^*$: range$(\mathbf{X}^*) \subseteq$ range$(\mathbf{A}^*)$. To compute an orthonormal basis for the range, we apply the QR factorization to $\mathbf{Y}$ and $\mathbf{X}^*$, respectively, to obtain orthonormal matrices $\mathbf{Q} \in \mathbb{R}^{M\times k}$ and $\mathbf{P} \in \mathbb{R}^{N\times k}$ with the same span. Therefore, the following approximations hold:

$$\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A}, \quad \mathbf{A} \approx \mathbf{APP}^*, \quad \text{and} \quad \mathbf{A} \approx \mathbf{QQ}^*\mathbf{APP}^*.$$

Sketching provides a tractable way to control the relative error in this approximation by varying the target rank parameter r since the error tends to zero as r

increases. We restate a few lemmas from [17, 36] to show how the error depends on the parameters $k = 2r + 1$ and $s = 2k + 1$. For use in these results, we define the *jth tail energy* of a matrix $\mathbf{A}$ as

$$\tau_j(\mathbf{A}) := \min_{\mathrm{Rank}(\mathbf{B})<j} \|\mathbf{A} - \mathbf{B}\|_{\mathrm{F}} = \left(\sum_{i \geq j} \sigma_i^2(\mathbf{A}) \right)^{\frac{1}{2}}.$$

LEMMA 3.2 (Theorem 10.5 in [17], Lemma A.5 in [36]). *Let $\mathbf{A} \in \mathbb{R}^{M \times N}$ be a matrix, and let r be the target rank parameter. Choose sketch parameters $k = 2r + 1$ and $s = 2k + 1$. Draw $\mathbf{\Upsilon}$, $\mathbf{\Omega}$ with i.i.d. standard normal entries, and compute* $(\mathbf{Q}, \mathbf{P})$ *according to* (3.1)–(3.3). *Then*

$$\begin{aligned} \mathbf{E}_{\mathbf{\Omega}} \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\mathrm{F}} &\leq \sqrt{2} \cdot \tau_{r+1}(\mathbf{A}), \\ \mathbf{E}_{\mathbf{\Upsilon}} \|\mathbf{A} - \mathbf{A}\mathbf{P}\mathbf{P}^*\|_{\mathrm{F}} &\leq \sqrt{2} \cdot \tau_{r+1}(\mathbf{A}), \\ \mathbf{E}_{\mathbf{\Omega},\mathbf{\Upsilon}} \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{P}\mathbf{P}^*\|_{\mathrm{F}} &\leq \sqrt{3} \cdot \tau_{r+1}(\mathbf{A}). \end{aligned}$$

We can form a low-rank approximation $\mathbf{A}$ by applying either $\mathbf{Q}$ or $\mathbf{P}^*$ (or both) to $\mathbf{A}$ to form the low-rank approximations $\mathbf{Q}(\mathbf{Q}^*\mathbf{A})$, $(\mathbf{A}\mathbf{P})\mathbf{P}^*$, or $\mathbf{Q}(\mathbf{Q}^*\mathbf{A}\mathbf{P})\mathbf{P}^*$ [35, 36]. However, we must access the matrix $\mathbf{A}$ again to compute these approximations. In the context of dynamic optimization, this second access to $\mathbf{A}$ is quite expensive, as it entails either storing the entire state matrix or solving the state equation again.

Instead, it is possible to infer the action of $\mathbf{A}$ from the sketches themselves. One approach, proposed in [36], uses the additional sketch $\mathbf{Z}$ to record the action of $\mathbf{A}$ on the random DRMs $\mathbf{\Phi}$ and $\mathbf{\Psi}$. We obtain an approximation of the core $\mathbf{Q}^*\mathbf{A}\mathbf{P}$ as

$$\begin{aligned} \mathbf{Z} &= \mathbf{\Phi}\mathbf{A}\mathbf{\Psi}^* \approx (\mathbf{\Phi}\mathbf{Q})(\mathbf{Q}^*\mathbf{A}\mathbf{P})(\mathbf{P}^*\mathbf{\Psi}^*), \\ \mathbf{C} &:= (\mathbf{\Phi}\mathbf{Q})^\dagger \mathbf{Z}((\mathbf{\Psi}\mathbf{P})^\dagger)^* \approx \mathbf{Q}^*\mathbf{A}\mathbf{P}. \end{aligned}$$

The following lemma quantifies the error in the core approximation.

LEMMA 3.3 (Lemma A.4 in [36]). *Let $\mathbf{A} \in \mathbb{R}^{M \times N}$ be a matrix, and let r be the target rank parameter. Choose sketch parameters $k = 2r + 1$ and $s = 2k + 1$. Draw $\mathbf{\Upsilon}, \mathbf{\Omega}, \mathbf{\Phi}, \mathbf{\Psi}$ with i.i.d. standard normal entries, and compute* $(\mathbf{Q}, \mathbf{P}, \mathbf{C})$ *according to* (3.1)–(3.4). *Then $\mathbf{C}$ is an unbiased estimate for the core matrix $\mathbf{Q}^*\mathbf{A}\mathbf{P}$ and*

$$\mathbf{E}_{\mathbf{\Phi},\mathbf{\Psi}} \|\mathbf{C} - \mathbf{Q}^*\mathbf{A}\mathbf{P}\|_{\mathrm{F}} \leq \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{P}\mathbf{P}^*\|_{\mathrm{F}}.$$

To compute the error in the final approximation $\{\{\mathbf{A}\}\}_r := \mathbf{Q}\mathbf{C}\mathbf{P}^*$, we note that

$$\begin{aligned} \|\mathbf{A} - \mathbf{Q}\mathbf{C}\mathbf{P}^*\|_{\mathrm{F}}^2 &= \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{P}\mathbf{P}^* + \mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{P}\mathbf{P}^* - \mathbf{Q}\mathbf{C}\mathbf{P}^*\|_{\mathrm{F}}^2 \\ &= \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\mathbf{P}\mathbf{P}^*\|_{\mathrm{F}}^2 + \|\mathbf{Q}(\mathbf{Q}^*\mathbf{A}\mathbf{P} - \mathbf{C})\mathbf{P}^*\|_{\mathrm{F}}^2. \end{aligned}$$

Thus we can combine Lemmas 3.2 and 3.3 to obtain the total reconstruction error.

THEOREM 3.4 (total reconstruction error [36]). *Let $\mathbf{A} \in \mathbb{R}^{M \times N}$ be a matrix, and let r be the target rank parameter. Choose sketch parameters $k = 2r + 1$ and $s = 2k + 1$. Compute range, co-range, and core sketches* $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ *according to* (3.2). *The low-rank approximation $\{\{\mathbf{A}\}\}_r$ computed in* (3.3)–(3.5) *satisfies*

$$\mathbf{E}_{\mathbf{\Omega},\mathbf{\Upsilon},\mathbf{\Phi},\mathbf{\Psi}} \|\mathbf{A} - \{\{\mathbf{A}\}\}_r\|_{\mathrm{F}} \leq \sqrt{6} \cdot \tau_{r+1}(\mathbf{A}).$$

This result shows that the rank-k approximation to $\mathbf{A}$ computed by sketching is only a constant factor farther from $\mathbf{A}$ than the best rank-r approximation on average. We shall later demonstrate how these error bounds can be used for a posteriori error estimation of the inexact gradient for dynamic optimization. It is also possible to obtain an unbiased estimate of the error in approximation, $\|\mathbf{A} - \{\{\mathbf{A}\}\}_r\|_{\mathrm{F}}$, in one streaming pass over the target matrix $\mathbf{A}$ using an additional test matrix; see [36, section 6] for details.

In this paper, we state most bounds in terms of the expected error. It is also possible to bound the probability of large deviations. Indeed, the probability that $\|\mathbf{A} - \{\{\mathbf{A}\}\}_r\|_{\mathrm{F}}$ deviates by ϵ from its expectation decreases superexponentially as the sketch parameters k and s increase since Lipschitz functions of Gaussian random variables exhibit dimension-free concentration. See [17, Theorem 10.9] for more details.

4. Randomized sketching for dynamic optimization.

This section presents our limited-memory algorithm to solve the dynamic optimization problem (2.2). Any first-order optimization method relies on the gradient of the objective function, so we begin with a discussion of how to compute a limited-memory approximate gradient in subsection 4.1. We also discuss how the same approach can be extended to apply the Hessian to a vector using limited memory. This enables usage of second-order methods. We next quantify the error in the approximate gradient. To quantify this error, we rely on regularity assumptions detailed in subsection 4.2. This analysis undergirds our results on the optimization algorithms presented in the next two subsections. In subsection 4.3, we present our first approach that considers computing the gradient using a fixed-rank sketch. This method has the advantage that it uses a fixed storage budget. However, for this method to work well, the state corresponding to any control must be well approximated by a fixed-rank sketch whose rank is known in advance. In subsection 4.4, we present our second approach, an adaptive method that updates the sketch rank to control the error in the gradient. We obtain a provably convergent optimization method by using this adaptive approach to compute the gradient within a trust-region algorithm. Unlike the fixed-rank method, this approach does not require a rank estimate a priori. However, this approach has the disadvantage that the storage budget required is dictated by the progress of the optimization algorithm and is not known a priori.

4.1. Computing first- and second-order information with limited memory.

To compute the gradient and to apply the Hessian with limited memory, we can sketch the state while solving the state equation, $c(\mathbf{U}, \mathbf{Z}) = \mathbf{0}$. Upon solving the adjoint equation, we reconstruct from the sketch to compute an approximate state. This allows us to compute an approximate gradient based on the approximate state.

4.1.1. Solving the state equation.

Fix the target rank parameter r. To set notation, denote by $\mathtt{mat}(\mathbf{U}, M, N)$ the state vectors at different time steps $\mathbf{u}_n$ collected into a matrix, $\mathtt{mat}(\mathbf{U}, M, N) := [\mathbf{u}_1 \cdots \mathbf{u}_N] \in \mathbb{R}^{M \times N}$. With some abuse of notation, we define the approximate state to be the low-rank approximation reconstructed via sketching the state matrix $\{\{\mathbf{U}\}\}_r := \{\{\mathtt{mat}(\mathbf{U}, M, N)\}\}_r$. Since the state matrix is computed forward in time starting from $\mathbf{u}_1$ to $\mathbf{u}_N$, we can simultaneously update the sketch matrices $\mathbf{X}, \mathbf{Y}$, and $\mathbf{Z}$ using the ColumnUpdate! function of the sketch class Algorithm A.1. The reduced objective function can be simultaneously exactly evaluated in this procedure. This method is presented as Algorithm 4.1. Here, the notation Function!() denotes a method that can modify its arguments or associated

Algorithm 4.1 Solve state equation, and compute exact objective function value.

```
Input: A control iterate $\mathbf{Z} \in \mathbb{R}^{m\times N}$, sketch object $\{\mathbf{U}\}_r$ for state and sketch rank
    parameter $r \le \min\{M, N\}$
Output: Updated sketch object $\{\mathbf{U}\}_r$ and reduced objective function value $F(\mathbf{Z})$
Storage: $\mathcal{O}(r(M+N)+mN)$
1: function SolveState!($\{\mathbf{U}\}_r$, $\mathbf{Z}$)
2:     $(\mathbf{u}_{\text{curr}}, F) \leftarrow (\mathbf{u}_0, 0)$
3:     for $n \leftarrow 1$ to $N$ do
4:         Solve $c_n(\mathbf{u}_{\text{curr}}, \mathbf{u}_{\text{next}}, \mathbf{z}_n) = 0$ for $\mathbf{u}_{\text{next}}$     Solve $n$th state equation
5:         $F \leftarrow F + f_n(\mathbf{u}_{\text{curr}}, \mathbf{u}_{\text{next}}, \mathbf{z}_n)$     Update objective function value
6:         $\{\mathbf{U}\}_r$.ColumnUpdate!$(\mathbf{u}_{\text{next}}, n)$     Update sketch with $n$th column of state
7:         $\mathbf{u}_{\text{curr}} \leftarrow \mathbf{u}_{\text{next}}$
8:     return $F$
```

class. In the context of the approximate state, we shall refer to $c(\mathbf{U}, \mathbf{Z})$ as the state residual.

4.1.2. Computing an approximate gradient from the sketched state. For a fixed control $\mathbf{Z}$, the true adjoint $\bar{\mathbf{\Lambda}}$ solves the adjoint equation (2.4) at the true state $\bar{\mathbf{U}}$, while the approximate adjoint $\widehat{\mathbf{\Lambda}}_r$ solves the adjoint equation (2.4) at the sketched state $\widehat{\mathbf{U}}_r = \{\{\bar{\mathbf{U}}\}\}_r$:

$$(\mathsf{d}_1 c(\widehat{\mathbf{U}}_r, \mathbf{Z}))^* \widehat{\mathbf{\Lambda}}_r = -\mathsf{d}_1 f(\widehat{\mathbf{U}}_r, \mathbf{Z}).$$

By analogy with the state residual $c(\mathbf{U}, \mathbf{Z})$, define the adjoint residual $h : \mathfrak{U}\times\mathfrak{U}\times\mathfrak{Z} \to \mathfrak{U}$ as

$$h(\mathbf{\Lambda}, \mathbf{U}, \mathbf{Z}) := \mathsf{d}_1 f(\mathbf{U}, \mathbf{Z}) + (\mathsf{d}_1 c(\mathbf{U}, \mathbf{Z}))^* \mathbf{\Lambda}.$$

The adjoint residual evaluated at arguments $(\mathbf{\Lambda}, \mathbf{U}, \mathbf{Z})$ is zero when $\mathbf{\Lambda}$ solves the adjoint equation (2.4) for any control $\mathbf{Z}$ and state $\mathbf{U}$. Consider in particular the special cases of this equality using the true state $h(\bar{\mathbf{\Lambda}}, \bar{\mathbf{U}}, \mathbf{Z}) = 0$ and using the sketched state $h(\widehat{\mathbf{\Lambda}}_r, \widehat{\mathbf{U}}_r, \mathbf{Z}) = 0$. For arbitrary $\mathbf{\Lambda}$, $\mathbf{U}$, and $\mathbf{Z}$, we define the map $g : \mathfrak{U}\times\mathfrak{U}\times\mathfrak{Z} \to \mathfrak{Z}$ as

$$g(\mathbf{\Lambda}, \mathbf{U}, \mathbf{Z}) := \mathsf{d}_2 f(\mathbf{U}, \mathbf{Z}) + (\mathsf{d}_2 c(\mathbf{U}, \mathbf{Z}))^* \mathbf{\Lambda}.$$

This function computes the gradient at $\mathbf{Z}$ when we use the true state $\bar{\mathbf{U}}$ and true adjoint $\bar{\mathbf{\Lambda}}$:

$$g(\bar{\mathbf{\Lambda}}, \bar{\mathbf{U}}, \mathbf{Z}) = \mathsf{d}_2 f(\bar{\mathbf{U}}, \mathbf{Z}) + (\mathsf{d}_2 c(\bar{\mathbf{U}}, \mathbf{Z}))^* \bar{\mathbf{\Lambda}} = \nabla F(\mathbf{Z}).$$

On the other hand, the function g can approximate the gradient using the sketched variables as

$$g_r(\mathbf{Z}) := g(\widehat{\mathbf{\Lambda}}_r, \widehat{\mathbf{U}}_r, \mathbf{Z}).$$

Algorithm 4.2 describes a backward-in-time procedure for computing a limited-memory approximate gradient $g_r(\mathbf{Z})$ from the sketched state $\widehat{\mathbf{U}}_r$.

4.2. Regularity assumptions. Throughout the remainder of the paper, we make the following regularity assumptions. These assumptions allow us to develop

Algorithm 4.2 Compute gradient from sketched state.

```
Input: A control iterate $\mathbf{Z} \in \mathbb{R}^{m\times N}$ and sketch object $\{\mathbf{U}\}_r$ for state
Output: Approximate gradient $g = g_r(\mathbf{Z}) \approx \nabla F(\mathbf{Z})$
Storage: $\mathcal{O}(r(M+N)+mN)$
 1: function GRADIENT($\{\mathbf{U}\}_r$, $\mathbf{Z}$)
 2:     $(\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{next}}) \leftarrow (\{\mathbf{U}\}_r.\texttt{Column}(N-1), \{\mathbf{U}\}_r.\texttt{Column}(N))$
 3:     Solve the adjoint equation at index $N$ for $\widehat{\boldsymbol{\lambda}}_{\mathrm{next}}$,
            $(\mathsf{d}_2 c_N(\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{next}}, \mathbf{z}_N))^* \widehat{\boldsymbol{\lambda}}_{\mathrm{next}} = \mathsf{d}_2 f_N(\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{next}}, \mathbf{z}_N)$
 4:     Compute gradient at index $N$,
            $g_N \leftarrow \mathsf{d}_3 f_N(\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{next}}, \mathbf{z}_N) + (\mathsf{d}_3 c_N(\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{next}}, \mathbf{z}_N))^* \widehat{\boldsymbol{\lambda}}_{\mathrm{next}}$
 5:     for $n \leftarrow N-1$ to 1 do
 6:         if $n = 1$ then
 7:             $\widehat{\mathbf{u}}_{\mathrm{prev}} \leftarrow \mathbf{u}_0$
 8:         else
 9:             $\widehat{\mathbf{u}}_{\mathrm{prev}} \leftarrow \{\mathbf{U}\}_r.\texttt{Column}(n-1))$
10:         Solve the adjoint equation at index $n$ for $\widehat{\boldsymbol{\lambda}}_{\mathrm{curr}}$,
            $(\mathsf{d}_2 c_n(\widehat{\mathbf{u}}_{\mathrm{prev}}, \widehat{\mathbf{u}}_{\mathrm{curr}}, \mathbf{z}_n))^* \widehat{\boldsymbol{\lambda}}_{\mathrm{curr}} = \mathsf{d}_2 f_n(\widehat{\mathbf{u}}_{\mathrm{prev}}, \widehat{\mathbf{u}}_{\mathrm{curr}}, \mathbf{z}_n) + \mathsf{d}_1 f_{n+1}(\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{next}}, \mathbf{z}_{n+1})$
                $- (\mathsf{d}_1 c_{n+1}(\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{next}}, \mathbf{z}_{n+1}))^* \widehat{\boldsymbol{\lambda}}_{\mathrm{next}}$
11:         Compute gradient at index $n$,
            $g_n \leftarrow \mathsf{d}_3 f_n(\widehat{\mathbf{u}}_{\mathrm{prev}}, \widehat{\mathbf{u}}_{\mathrm{curr}}, \mathbf{z}_n) + (\mathsf{d}_3 c_n(\widehat{\mathbf{u}}_{\mathrm{prev}}, \widehat{\mathbf{u}}_{\mathrm{curr}}, \mathbf{z}_n))^* \widehat{\boldsymbol{\lambda}}_{\mathrm{curr}}$
12:         $(\widehat{\mathbf{u}}_{\mathrm{next}}, \widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\boldsymbol{\lambda}}_{\mathrm{next}}) \leftarrow (\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{prev}}, \widehat{\boldsymbol{\lambda}}_{\mathrm{curr}})$
13:     return $g = [g_1, \ldots, g_N]$
```

provable guarantees on the optimization error in the algorithms presented in the next two subsections. These conditions are adapted from [41].

ASSUMPTION 1. *Assume that the following conditions hold for the dynamic optimization problem* (2.2)*:*

1. *The set of states corresponding to controls in an open and bounded set $\mathfrak{Z}_0 \subseteq \mathfrak{Z}$ is bounded: there exists $\mathfrak{U}_0 \subset \mathfrak{U}$ open and bounded such that $\{\mathbf{U} \in \mathfrak{U} \mid \exists\, \mathbf{Z} \in \mathfrak{Z}_0,\ c(\mathbf{U}, \mathbf{Z}) = \mathbf{0}\} \subseteq \mathfrak{U}_0$.*
2. *There exist singular value thresholds $0 < \sigma_0 \le \sigma_1 < \infty$ such that for any $\mathbf{U} \in \mathfrak{U}_0$ and $\mathbf{Z} \in \mathfrak{Z}_0$, the state Jacobian matrix $\mathsf{d}_1 c(\mathbf{U}, \mathbf{Z})$ satisfies $\sigma_0 \le \sigma_{\min}(\mathsf{d}_1 c(\mathbf{U}, \mathbf{Z})) \le \sigma_{\max}(\mathsf{d}_1 c(\mathbf{U}, \mathbf{Z})) \le \sigma_1$.*
3. *The following functions are Lipschitz continuous on $\mathfrak{U}_0 \times \mathfrak{Z}_0$ with respect to their first arguments, and their respective Lipschitz moduli are independent of $Z \in \mathfrak{Z}_0$:*
   (a) *the state Jacobian of the constraint, $\mathsf{d}_1 c(\mathbf{U}, \mathbf{Z})$;*
   (b) *the control Jacobian of the constraint, $\mathsf{d}_2 c(\mathbf{U}, \mathbf{Z})$;*
   (c) *the state gradient of the objective function, $\mathsf{d}_1 f(\mathbf{U}, \mathbf{Z})$;*
   (d) *the control gradient of the objective function, $\mathsf{d}_2 f(\mathbf{U}, \mathbf{Z})$.*

These assumptions are often satisfied in applications. For example, we show in section 5 that they hold for optimal control problems with parabolic PDE constraints. We note that the existence of σ_1 in Assumption 1 is satisfied because $\mathfrak{U}_0$ is bounded and $\mathsf{d}_1 c(\mathbf{U},\mathbf{Z})$ is Lipschitz continuous with respect to its first argument with Lipschitz modulus K_1, which is independent of $Z \in \mathfrak{Z}_0$. In particular, we have that

$$\frac{\|\mathsf{d}_1 c(\mathbf{U},\mathbf{Z})\mathbf{H}\|_2}{\|\mathbf{H}\|_2} = \frac{1}{\|\mathbf{H}\|_2}\left\|\lim_{t\downarrow 0}\frac{c(\mathbf{U}+t\mathbf{H},\mathbf{Z})-c(\mathbf{U},\mathbf{Z})}{t}\right\|_2 \le K_1 \quad \forall\, \mathbf{H}\in\mathfrak{U}\setminus\{0\}.$$

Therefore, we can take $\sigma_1 = K_1$.

4.3. A fixed-rank approach. A natural limited-memory algorithm to solve the dynamic optimization problem (2.2) is to fix the sketch rank parameter r used to compute the gradient a priori. Algorithm 4.3 shows the steps involved in this method. The resulting approximate gradient can be used inside any first-order optimization method to (approximately) solve the dynamic optimization problem (2.2). One can also apply matrix-free second-order optimization methods such as Newton–Krylov methods. These methods require only the application of the Hessian to a vector: for an arbitrary vector $\mathbf{V} \in \mathfrak{Z}$, we must compute $\nabla^2 F(\mathbf{Z})\mathbf{V}$. Using the chain rule, we can apply the Hessian to $\mathbf{V}$ by first computing the *state sensitivity* $\bar{\mathbf{W}} := S'(\mathbf{Z})\mathbf{V} \in \mathfrak{U}$ (i.e., the solution to (2.3)) and then the *adjoint sensitivity* $\bar{\mathbf{P}} := \Lambda'(\mathbf{Z})\mathbf{V} \in \mathfrak{U}$. The state sensitivity is computed forward in time, while the adjoint sensitivity is computed backward in time. We can control the storage footprint for these operations by sketching the state, adjoint, and state sensitivity. Algorithms A.3 to A.6 detail the steps required to apply the Hessian to a vector with storage $\mathcal{O}((r_1+r_2+r_3)(M+N)+mN)$ for rank parameters $r_1, r_2, r_3 \le \min\{M,N\}$.

In this section, we analyze the error of the fixed-rank method and prove a useful stopping criterion under Assumption 1.

Algorithm 4.3 Fixed-rank algorithm for approximate gradient.

Input: A control iterate $\mathbf{Z} \in \mathbb{R}^{m\times N}$ and rank parameter $r \le \min\{M,N\}$.
Output: Approximate gradient $g_r(\mathbf{Z})$
Storage: $\mathcal{O}(r(M+N)+mN)$

```
1: function FixedRankGradient($\mathbf{Z}$)
2:     $\{\mathbf{U}\}_r \leftarrow$ Sketch($M, N, \text{rank} = r$)      Instantiate sketch object for state
3:     $F \leftarrow$ SolveState!($\{\mathbf{U}\}_r, \mathbf{Z}$)          Solve state equation
4:     $\{\mathbf{U}\}_r$.Reconstruct!()                      Reconstruct low-rank factors
5:     $g \leftarrow$ Gradient($\{\mathbf{U}\}_r, \mathbf{Z}$)              Compute gradient
6:     return $g$
```

Recall that variables with bars $\bar{\mathbf{U}}$, $\bar{\mathbf{\Lambda}}$ denote the exact solutions to the state or adjoint equation uniquely determined for a fixed control $\mathbf{Z} \in \mathcal{Z}$, while variables with hats $\widehat{\mathbf{U}}_r$, $\widehat{\mathbf{\Lambda}}_r$ are approximate solutions with approximation error controlled by the rank parameter r.

Proposition 4.1. *Suppose Assumption* 1 *holds for a bounded control set* $\mathfrak{Z}_0$. *Then there exist* κ_0, $\kappa_1 > 0$ *such that the error in the state satisfies*

$$\kappa_0\|\mathbf{U}-\bar{\mathbf{U}}\|_2 \le \|c(\mathbf{U},\mathbf{Z})\|_2 \le \kappa_1\|\mathbf{U}-\bar{\mathbf{U}}\|_2 \quad \forall\,\mathbf{U}\in\mathfrak{U}_0,\ \mathbf{Z}\in\mathfrak{Z}_0, \tag{4.1}$$

where $\mathfrak{U}_0 \subseteq \mathfrak{U}$ *is defined in condition* 1 *in Assumption* 1.

Furthermore, the error in the adjoint is controlled by the adjoint residual together with the state residual: for some $\kappa_2 > 0$ and $\kappa_3 > 0$,

$$\|\mathbf{\Lambda} - \bar{\mathbf{\Lambda}}\|_2 \le \kappa_2 \|c(\mathbf{U}, \mathbf{Z})\|_2 + \kappa_3 \|h(\mathbf{\Lambda}, \mathbf{U}, \mathbf{Z})\|_2 \quad \forall\, \mathbf{U},\, \mathbf{\Lambda} \in \mathfrak{U}_0, \quad \forall\, \mathbf{Z} \in \mathfrak{Z}_0. \tag{4.2}$$

Hence the error in the gradient is controlled by the adjoint and state residuals: for some $\kappa_4 > 0$ and $\kappa_5 > 0$,

$$\begin{aligned} \|g(\mathbf{\Lambda}, \mathbf{U}, \mathbf{Z}) - g(\bar{\mathbf{\Lambda}}, \bar{\mathbf{U}}, \mathbf{Z})\|_2 &= \|g(\mathbf{\Lambda}, \mathbf{U}, \mathbf{Z}) - \nabla F(\mathbf{Z})\|_2 \\ &\le \kappa_4 \|c(\mathbf{U}, \mathbf{Z})\|_2 + \kappa_5 \|h(\mathbf{\Lambda}, \mathbf{U}, \mathbf{Z})\|_2. \end{aligned} \tag{4.3}$$

Remark 4.2. All constants in Proposition 4.1 depend only on finite quantities defined by Assumption 1.

Proof. The proof of this result is similar to the proofs of Propositions A.1–A.2 in [41]. To bound the error in the state, recall that the state residual is zero when evaluated at the true state, $c(\bar{\mathbf{U}}, \mathbf{Z}) = 0$. Therefore,

$$c(\mathbf{U}, \mathbf{Z}) = c(\mathbf{U}, \mathbf{Z}) - c(\bar{\mathbf{U}}, \mathbf{Z}) = \int_0^1 \mathsf{d}_1 c(\bar{\mathbf{U}} + t(\mathbf{U} - \bar{\mathbf{U}}), \mathbf{Z}) \cdot (\mathbf{U} - \bar{\mathbf{U}})\, dt.$$

The error bound (4.1) then follows from Assumption 1.2 using $\kappa_0 = \sigma_0$ and $\kappa_1 = \sigma_1$. Similarly, we show the bound on the adjoint error using the adjoint residual,

$$h(\mathbf{\Lambda}, \bar{\mathbf{U}}, \mathbf{Z}) = h(\mathbf{\Lambda}, \bar{\mathbf{U}}, \mathbf{Z}) - h(\bar{\mathbf{\Lambda}}, \bar{\mathbf{U}}, \mathbf{Z}) = (\mathsf{d}_1 c(\bar{\mathbf{U}}, \mathbf{Z}))^* (\mathbf{\Lambda} - \bar{\mathbf{\Lambda}}),$$

together with the Cauchy–Schwarz inequality and Assumption 1.2 to see that

$$\sigma_0 \|\mathbf{\Lambda} - \bar{\mathbf{\Lambda}}\|_2 \le \|h(\mathbf{\Lambda}, \bar{\mathbf{U}}, \mathbf{Z})\|_2 \le \sigma_1 \|\mathbf{\Lambda} - \bar{\mathbf{\Lambda}}\|_2.$$

We now bound the adjoint residual as

$$\begin{aligned} \|h(\mathbf{\Lambda}, \bar{\mathbf{U}}, \mathbf{Z})\|_2 &\le \|h(\mathbf{\Lambda}, \mathbf{U}, \mathbf{Z})\|_2 + \|h(\mathbf{\Lambda}, \bar{\mathbf{U}}, \mathbf{Z}) - h(\mathbf{\Lambda}, \mathbf{U}, \mathbf{Z})\|_2 \\ &\le \|h(\mathbf{\Lambda}, \mathbf{U}, \mathbf{Z})\|_2 + \|\mathsf{d}_1 c(\mathbf{U}, \mathbf{Z}) - \mathsf{d}_1 c(\bar{\mathbf{U}}, \mathbf{Z})\|_{\mathrm{F}} \|\Lambda\|_2 \\ &\quad + \|\mathsf{d}_1 f(\mathbf{U}, \mathbf{Z}) - \mathsf{d}_1 f(\bar{\mathbf{U}}, \mathbf{Z})\|_2. \end{aligned}$$

The bound (4.2) follows from the Lipschitz continuity of $\mathsf{d}_1 c$ and $\mathsf{d}_1 f$, the boundedness of $\mathfrak{U}_0 \times \mathfrak{Z}_0$, and (4.1). The proof of (4.3) is identical to the proof of Proposition A.2 in [41]. In particular, (4.3) follows from (4.1), (4.2) and the assumed Lipschitz continuity of $\mathsf{d}_2 c$ and $\mathsf{d}_2 f$. ∎

COROLLARY 4.3. *Suppose Assumption 1 holds for a bounded control set $\mathfrak{Z}_0$. Fix a control $\mathbf{Z} \in \mathfrak{Z}_0$ and rank parameter r. Suppose the approximate state $\widehat{\mathbf{U}}_r = \{\{\bar{\mathbf{U}}\}\}_r$ is in $\mathfrak{U}_0$ almost surely. Then the state residual is bounded by the tail energy of the true state $\bar{\mathbf{U}}$ on average:*

$$\mathbf{E}\, \|c(\widehat{\mathbf{U}}_r, \mathbf{Z})\|_2 \le \sqrt{6}\kappa_1\, \tau_{r+1}(\mathtt{mat}(\bar{\mathbf{U}}, M, N)).$$

Now recall that the approximate adjoint $\widehat{\mathbf{\Lambda}}_r$ solves the adjoint equation (2.4) *at the approximate state $\widehat{\mathbf{U}}_r$. Suppose that $\widehat{\mathbf{\Lambda}}_r \in \mathfrak{U}_0$ almost surely. Then the error in the adjoints satisfies*

$$\mathbf{E}\, \|\widehat{\mathbf{\Lambda}}_r - \bar{\mathbf{\Lambda}}\|_2 \le \sqrt{6}\kappa_1\kappa_2\, \tau_{r+1}(\mathtt{mat}(\bar{\mathbf{U}}, M, N)). \tag{4.4}$$

Finally, the approximate gradient $g_r(\mathbf{Z}) = g(\widehat{\mathbf{\Lambda}}_r, \widehat{\mathbf{U}}_r, \mathbf{Z})$ satisfies the error bound

$$\mathbf{E}\, \|g_r(\mathbf{Z}) - \nabla F(\mathbf{Z})\|_2 \le \sqrt{6}\kappa_1\kappa_4\, \tau_{r+1}(\mathtt{mat}(\bar{\mathbf{U}}, M, N)). \tag{4.5}$$

Proof. This result is a direct consequence of Proposition 4.1 and Theorem 3.4. ☐

Corollary 4.3 suggests that we should choose the fixed-rank parameter r so that the tail energy, $\tau_{r+1}(\mathtt{mat}(\bar{\mathbf{U}}, M, N))$, is small. However, it can be difficult to choose a good fixed-rank parameter in advance since the tail energy of the true state $\bar{\mathbf{U}}$ depends on the control variable $\mathbf{Z}$. Under stronger assumptions on the reduced objective F, we can bound the distance from a given control to the optimum as a function of the approximate gradient and the state residual. Both of these are easy to compute, and hence this result can be used as a stopping criterion.

THEOREM 4.4. *Suppose Assumption* 1 *holds for a bounded control set $\mathfrak{Z}_0$. Fix a control $\mathbf{Z} \in \mathfrak{Z}_0$ and rank parameter r, and suppose the approximate state $\widehat{\mathbf{U}}_r$ and adjoint $\widehat{\mathbf{\Lambda}}_r$ are in $\mathfrak{U}_0$ almost surely. Additionally assume that the reduced objective function F is strongly convex on $\mathfrak{Z}_0$ with parameter $\alpha > 0$. Let $\mathbf{Z}^\star \in \mathfrak{Z}_0$ denote the solution to the reduced dynamic optimization problem* (2.2). *Then*

$$\alpha\|\mathbf{Z} - \mathbf{Z}^\star\|_2 \le \kappa_4 \|c(\mathbf{U}_r, \mathbf{Z})\|_2 + \|g_r(\mathbf{Z})\|_2. \tag{4.6}$$

Proof. Using the strong convexity of F and the optimality of $\mathbf{Z}^\star$, the error in control is bounded above by the gradient of the reduced objective function F as

$$\alpha\|\mathbf{Z} - \mathbf{Z}^\star\|_2^2 \le \langle \nabla F(\mathbf{Z}) - \nabla F(\mathbf{Z}^\star), \mathbf{Z} - \mathbf{Z}^\star\rangle_{\mathfrak{Z}} = \langle \nabla F(\mathbf{Z}), \mathbf{Z} - \mathbf{Z}^\star\rangle_{\mathfrak{Z}}.$$

Applying the Cauchy–Schwarz inequality and employing (4.3) ensures that

$$\begin{aligned}\alpha\|\mathbf{Z} - \mathbf{Z}^\star\|_2 &\le \|\nabla F(\mathbf{Z}) - g_r(\mathbf{Z}) + g_r(\mathbf{Z})\|_2 \\ &\le \|\nabla F(\mathbf{Z}) - g_r(\mathbf{Z})\|_2 + \|g_r(\mathbf{Z})\|_2 \\ &\le \kappa_4 \|c(\mathbf{U}_r, \mathbf{Z})\|_2 + \|g_r(\mathbf{Z})\|_2. \end{aligned}$$ ☐

To use Theorem 4.4, run any optimization method using the approximate gradient $g_r(\mathbf{Z})$. Suppose the method terminates after ℓ iterations at control $\mathbf{Z}^{(\ell)}$ so that $\|g_r(\mathbf{Z}^{(\ell)})\|_2 \le \epsilon$. Theorem 4.4 shows that the error in our optimal control is controlled by the state residual:

$$\alpha\|\mathbf{Z}^{(\ell)} - \mathbf{Z}^\star\|_2 \le \kappa_4 \|c(\mathbf{U}_r, \mathbf{Z}^{(\ell)})\|_2 + \epsilon.$$

4.4. An adaptive-rank approach. In this section, we introduce an optimization algorithm, the *sketched trust-region method*, that dynamically adjusts the sketching rank parameter used to compute the approximate gradient. The rank is chosen to guarantee convergence to a stationary point of the dynamic optimization problem (2.2). This algorithm relies on the trust-region framework [8], which converges despite inexact first- and second-order information [19, 24, 25]. Unlike the fixed-rank method described in the previous section, the sketched trust-region method is a complete limited-memory optimization recipe.

Let us describe the standard trust-region method and the conditions required for convergence in the context of the dynamic optimization problem (2.2). Let $\mathbf{Z}^{(\ell)}$ be the control at the ℓth iteration, with corresponding reduced objective function value $f^{(\ell)} := F(\mathbf{Z}^{(\ell)})$. The trust-region method approximates the reduced objective function centered around $\mathbf{Z}^{(\ell)}$, $\boldsymbol{\nu} \mapsto F(\mathbf{Z}^{(\ell)} + \boldsymbol{\nu})$, by a quadratic model

$$m^{(\ell)}(\boldsymbol{\nu}) := f^{(\ell)} + \langle g^{(\ell)}, \boldsymbol{\nu}\rangle_{\mathfrak{Z}} + \frac{1}{2}\langle H^{(\ell)}\boldsymbol{\nu}, \boldsymbol{\nu}\rangle_{\mathfrak{Z}}.$$

To find the next iterate, the trust-region method computes a step $\bar{\boldsymbol{\nu}}$ which approximately[1] solves the trust-region subproblem constrained by the trust-region radius $\Delta^{(\ell)}$:

$$\begin{array}{ll} \text{minimize} & m^{(\ell)}(\boldsymbol{\nu}) \\ \text{subject to} & \|\boldsymbol{\nu}\|_2 \le \Delta^{(\ell)}. \end{array} \tag{4.8}$$

This step is accepted so long as the actual decrease in the objective function value is large enough relative to the predicted decrease according to the model $m^{(\ell)}$. If the step is accepted and the actual reduction exceeds a specified threshold, the trust-region radius $\Delta^{(\ell)}$ is increased. If the step is rejected, we decrease the trust-region radius.

To ensure global convergence of the trust-region method, the model used to form the trust-region subproblem must satisfy Assumption 2 [19, 24].

Assumption 2 (trust-region model).

1. *The approximate gradient $g^{(\ell)}$ is close to the true gradient $\nabla F(\mathbf{Z}^{(\ell)})$ in that it satisfies*

$$\|g^{(\ell)} - \nabla F(\mathbf{Z}^{(\ell)})\|_2 \le \theta \min\left\{\|g^{(\ell)}\|_2, \Delta^{(\ell)}\right\} \tag{4.9}$$

   for some fixed $\theta > 0$ independent of ℓ.
2. *The approximate Hessians $H^{(\ell)}$ are bounded independent of k: there exists $\tau_1 > 0$ such that*

$$\|H^{(\ell)}\|_{\mathrm{F}} \le \tau_1 < \infty \quad \forall\, \ell = 1, 2, \dots .$$

We will show below how to ensure the first requirement with an approximate gradient $g^{(\ell)} := g_r(\mathbf{Z}^{(\ell)})$ computed using the sketched state $\widehat{\mathbf{U}}_r$ with a sufficiently large rank parameter r. The second requirement is easily ensured by setting $H^{(\ell)}$ to be the identity, while we expect (and observe) faster convergence in practice when $H^{(\ell)}$ is the approximate Hessian. See Algorithm A.5, which shows how to apply the approximate Hessian. Convergence is guaranteed regardless of the rank chosen for the Hessian approximation. We suggest fixing this parameter to be the same as the rank parameter for the approximate gradient.

4.4.1. Choosing the rank to guarantee convergence.

The sketched trust-region method sets $g^{(\ell)} = g_r(\mathbf{Z}^{(\ell)})$ for some rank r. Algorithm 4.4 ensures that r is chosen large enough that this approximate gradient satisfies the error bound (4.9), as proved in the following lemma. The function $\mu : \mathbb{N} \times [0, \infty) \to \mathbb{N}$ on line 9 of Algorithm 4.4 dictates how the rank r is increased and therefore is increasing in its first argument and decreasing in its second. A simple choice would be $\mu(r, \tau) = 2r$ or $\mu(r, \tau) = r + 1$. The function `ResidualNorm`, listed in Algorithm A.2, computes the Euclidean norm of the state residual.

Lemma 4.5. *Instate Assumption 1. Compute the gradient approximation $g^{(\ell)}$ using the adaptive-rank algorithm (Algorithm 4.4) with fixed state residual tolerance $\kappa_{grad} > 0$. Then $g^{(\ell)}$ satisfies the gradient error bound (4.9) with $\theta = \kappa_4 \kappa_{grad}$.*

[1]Formally, the algorithm computes a step $\bar{\boldsymbol{\nu}}$ that satisfies the fraction of Cauchy decrease condition [8]:

$$m^{(\ell)}(\mathbf{0}) - m^{(\ell)}(\bar{\boldsymbol{\nu}}) \ge \kappa_{\mathrm{fcd}} \|g^{(\ell)}\|_2 \min\left\{\Delta^{(\ell)}, \frac{\|g^{(\ell)}\|_2}{1 + \|H^{(\ell)}\|_{\mathrm{F}}}\right\} \tag{4.7}$$

for some $\kappa_{\mathrm{fcd}} \ge 0$ independent of ℓ. This condition is easy to achieve using, e.g., the dogleg or truncated conjugate gradient method to compute $\bar{\boldsymbol{\nu}}$.

Proof. The adaptive-rank algorithm controls the error in the gradient approximation by increasing the target rank parameter until the constraint residual satisfies

$$\|c(\widehat{\mathbf{U}}_r, \mathbf{Z}^{(\ell)})\|_2 \leq \kappa_{\text{grad}} \min\left\{\|g_r(\mathbf{Z}^{(\ell)})\|_2, \Delta^{(\ell)}\right\}. \tag{4.10}$$

A rank that satisfies (4.10) necessarily exists since the residual norm $\|c(\widehat{\mathbf{U}}_r, \mathbf{Z}^{(\ell)})\|_2 \to 0$ as $r \to \min\{M, N\}$. Therefore, Proposition 4.1 provides a bound on the error in the gradient approximation,

$$\|g_r(\mathbf{Z}^{(\ell)}) - \nabla F(\mathbf{Z}^{(\ell)})\|_2 \leq \kappa_4 \|c(\widehat{\mathbf{U}}_r, \mathbf{Z}^{(\ell)})\|_2 \leq \kappa_4 \kappa_{\text{grad}} \min\left\{\|g_r(\mathbf{Z}^{(\ell)})\|_2, \Delta^{(\ell)}\right\}. \qquad \square$$

Algorithm 4.4 Adaptive-rank algorithm for approximate gradient.

Input: A control iterate $\mathbf{Z} \in \mathbb{R}^{m \times N}$, initial rank estimate r, sketch object for state $\{\mathbf{U}\}_r$, trust-region radius $\Delta > 0$, state residual tolerance $\kappa_{\text{grad}} > 0$, and rank update function $\mu : \mathbb{N} \times [0, \infty) \to \mathbb{N}$.
Output: Approximate gradient $g_r(\mathbf{Z}) \approx \nabla F(\mathbf{Z})$ for rank parameter r such that the bound (4.10) is satisfied.
Storage: $\mathcal{O}(r(M + N) + mN)$ for some rank parameter $r \leq \min\{M, N\}$.

```
function AdaptiveRankGradient($\mathbf{Z}$, $r$, $\{\mathbf{U}\}_r$)
    repeat
        $\{\mathbf{U}\}_r$.Reconstruct!()                    Reconstruct low-rank factors
        rnorm ← ResidualNorm($\{\mathbf{U}\}_r, \mathbf{Z}$)      Compute norm of constraint residual
        $g$ ← Gradient($\{\mathbf{U}\}_r, \mathbf{Z}$)            Compute gradient
        rtol ← $\kappa_{\text{grad}} \cdot \min\{\|g\|_2, \Delta\}$.       Compute residual tolerance
        if rnorm ≤ rtol then                       Gradient approximation satisfies (4.9)
            return $g$
        $r$ ← $\mu(r, \text{rtol})$                          Increase rank parameter
        $\{\mathbf{U}\}_r$ ← Initialize!($M, N, \text{rank} = r$)   Re-initialize state sketch
        $F$ ← SolveState!($\{\mathbf{U}\}_r, \mathbf{Z}$).         Solve state equation
    until $r > \min\{M, N\}$
    return $g$,$r$
```

Remark 4.6. The fixed-rank algorithm (Algorithm 4.3) computes an approximate gradient with nondeterministic error bounded in expectation by the tail energy (4.5). The runtime of this algorithm is deterministic and depends on the rank parameter r. In contrast, the adaptive-rank algorithm (Algorithm 4.4) is guaranteed to produce an approximate gradient that satisfies the gradient error bound (4.9) with $\theta = \kappa_4 \kappa_{\text{grad}}$. However, the runtime is stochastic since it depends on the quality of the state sketch.

4.4.2. Sketched trust-region algorithm. We present the resulting sketched trust-region algorithm as Algorithm 4.5. To start the optimization, we use an initial trust-region radius $\Delta^{(0)}$, the initial rank parameter r_0, and the initial control $\mathbf{Z}^{(0)}$. The trust-region hyperparameters of Algorithm 4.5 are the ratio of reduction thresholds $0 < \eta_1 < \eta_2 < 1$ and the trust-region radius update parameter $\gamma \in (0, 1)$. The function SolveTRSubProblem computes the step $\bar{\boldsymbol{\nu}}$ that approximately solves the trust-region subproblem (4.8) and satisfies the fraction of Cauchy decrease condition (4.7). Internally, SolveTRSubProblem may use the function ApplyFixedRankHessian (see Algorithm A.5) to apply the fixed-rank Hessian approximation. To validate the trust-region step, we compare the actual and predicted reductions,

$$\text{ared}^{(\ell)} := F(\mathbf{Z}^{(\ell)}) - F(\mathbf{Z}^{(\ell)} + \bar{\boldsymbol{\nu}}) \quad \text{and} \quad \text{pred}^{(\ell)} := m^{(\ell)}(\mathbf{0}) - m^{(\ell)}(\bar{\boldsymbol{\nu}}).$$

We accept the step if their ratio is greater than the threshold η_1. The predicted reduction is readily computed as a byproduct of the trust-region subproblem solve; see, e.g., [8, Chap. 17.4], for more details. The actual reduction requires us to solve the state and evaluate the reduced objective function at the control candidate $\mathbf{Z}^{(\ell)}+\bar{\boldsymbol{\nu}}$. To do this, we apply Algorithm 4.1, which only requires the storage of two state time steps. Note that the sketched trust-region method sketches the state at $\mathbf{Z}^{(\ell)} + \bar{\boldsymbol{\nu}}$ so that (if the step is accepted) the approximate gradient can be computed using the sketch without solving the state equation again.

Algorithm 4.5 Sketched trust-region algorithm.

```
Input: Initial control $\mathbf{Z}^{(0)}$, trust-region radius $\Delta^{(0)}$, target rank parameter $r_0$,
       and trust-region hyperparameter set $P = \{\eta_1, \eta_2, \gamma\}$
Output: Control iterate $\mathbf{Z}^{(K)}$ such that the stopping criterion is satisfied
 1: function SketchedTrustRegion($\mathbf{Z}^{(0)}$, $\Delta^{(0)}$, $r_0$, $P$)
 2:     $\{\mathbf{U}\}_r \leftarrow$ Sketch($M, N, \text{rank} = r_0$)          Instantiate state sketch
 3:     $f^{(0)} \leftarrow$ SolveState!($\{\mathbf{U}\}_r, \mathbf{Z}^{(\ell)}$)          Sketch state and evaluate objective
 4:     $r \leftarrow r_0$                                                  Set sketch rank
 5:     while "Not Converged" do
 6:         $(g^{(\ell)}, r) \leftarrow$ AdaptiveRankGradient($\mathbf{Z}^{(\ell)}, r, \{\mathbf{U}\}_r$)   Approximate gradient
 7:         $(\bar{\boldsymbol{\nu}}, \text{pred}^{(\ell)}) \leftarrow$ SolveTRSubProblem($g^{(\ell)}, \Delta^{(\ell)}$)   Compute trial step
 8:         $\{\mathbf{U}\}_r$.Initialize!($M, N, \text{rank} = r$)          Re-initialize state sketch
 9:         $f^{(\ell+1)} \leftarrow$ SolveState!($\{\mathbf{U}\}_r, \mathbf{Z}^{(\ell)} + \bar{\boldsymbol{\nu}}$)   Compute new objective function value
10:         $\rho^{(\ell)} = (f^{(\ell)} - f^{(\ell+1)})/\text{pred}^{(\ell)}$          Compute ratio of reduction
11:         if $\rho^{(\ell)} \geq \eta_1$ then                                Validate step using ratio of reduction
12:             $\mathbf{Z}^{(\ell+1)} = \mathbf{Z}^{(\ell)} + \bar{\boldsymbol{\nu}}$
13:         else
14:             $\mathbf{Z}^{(\ell+1)} = \mathbf{Z}^{(\ell)}$
15:         if $\rho^{(\ell)} \geq \eta_2$ then                                Update trust-region radius
16:             $\Delta^{(\ell+1)} \in [\Delta^{(\ell)}, \infty)$
17:         else if $\rho^{(\ell)} \leq \eta_1$ then
18:             $\Delta^{(\ell+1)} \in (0, \gamma\|\bar{\boldsymbol{\nu}}\|_2]$
19:         else
20:             $\Delta^{(\ell+1)} \in [\gamma\|\bar{\boldsymbol{\nu}}\|_2, \Delta^{(\ell)}]$
```

In practical applications, it can be difficult to estimate a priori the rank of the state matrix for a given control. Algorithm 4.5 helpfully learns the rank while solving the optimization problem but could require full rank (and hence quite a lot of memory) in the worst case. If memory constraints are severe, we might recommend using the fixed-rank algorithm or checkpointing instead, or a hybrid approach that adaptively increases the rank up to some upper bound.

The following theorem shows that the sequence of iterates $\{\mathbf{Z}^{(\ell)}\}$ generated by Algorithm 4.5 converges to a stationary point of the reduced objective function F.

Theorem 4.7 (convergence of the sketched trust-region algorithm). *Instate Assumption* 1, *and further suppose that the reduced objective function F is bounded below and twice continuously differentiable with locally uniformly bounded Hessian: for any bounded convex set $\mathfrak{Z}_0 \subset \mathfrak{Z}$, there exists $\tau_0 > 0$ such that*

$$\|\nabla^2 F(\mathbf{Z})\|_{\text{F}} \leq \tau_0 < \infty \quad \forall\, \mathbf{Z} \in \mathfrak{Z}_0.$$

Suppose that the iterates $\mathbf{Z}^{(\ell)}$ *generated by Algorithm* 4.5 *lie in the open, bounded, and convex set* $\mathfrak{Z}_0 \subset \mathfrak{Z}$ *for all* ℓ. *Then the sequence of iterates* $\{\mathbf{Z}^{(\ell)}\}$ *satisfies*

$$\liminf_{\ell\to\infty} \|g_r^{(\ell)}\|_2 = \liminf_{\ell\to\infty} \|\nabla F(\mathbf{Z}^{(\ell)})\|_2 = 0.$$

Proof. Notice that the trust-region model used by Algorithm 4.5 satisfies Assumption 2. Therefore, the proof of this result follows from the convergence analysis for the inexact trust-region method in [24] with only a slight modification to account for the local assumptions associated with $\mathfrak{Z}_0$. ∎

Remark 4.8. The assumption that F is twice continuously differentiable can be relaxed to the requirement that F is continuously differentiable with Lipschitz continuous gradient. In this case, the proof of Theorem 4.7 is virtually identical to the proof in [24]; however, the proofs of Lemmas A.2 and A.3 in [24] must be updated accordingly.

Remark 4.9. Algorithm 4.5 converges deterministically in the sense of Theorem 4.7, although the *time* required for convergence is random since it uses the adaptive algorithm (Algorithm 4.4) to compute the search direction. Algorithm 4.4 may require multiple full state solves (cf. line 11) to obtain a gradient that satisfies (4.9). When the state is low rank, or when the rank parameter r estimates the rank well, very few additional solves will be necessary (depending on the rank update function μ). Moreover, the rank parameter never decreases in Algorithm 4.5; hence the per-iteration cost saturates after a finite number of iterations (again depending on μ).

5. Optimal control of linear parabolic PDEs.

In this section, we introduce a class of linear parabolic optimal control problems and discuss how to discretize them to obtain a problem of the form (2.1) that satisfies Assumption 1 and the inexact gradient condition (4.9). Let $\Omega \subset \mathbb{R}^d$ be an open, connected, and bounded set, and let $\Gamma \subseteq \Omega \cup \partial\Omega$, where $\partial\Omega$ denotes the boundary of Ω. The set Γ is the spatial support of the control function and permits both boundary and volumetric controls. The state is supported on the space-time cylinder $\Omega_T := (0,T)\times\Omega$, and the control is supported on $\Gamma_T := (0,T)\times\Gamma$ for $T > 0$. We denote by $H^1(\Omega)$ the usual Sobolev space of $L^2(\Omega)$-functions with weak derivatives in $L^2(\Omega)$ and let $V \subseteq H^1(\Omega)$ be a separable Hilbert space such that V is continuously and densely embedded into $L^2(\Omega)$ (typically, $V = H^1(\Omega)$ or $V = H_0^1(\Omega)$). Furthermore, we assume that Γ is sufficiently regular so that

$$v \mapsto \int_\Gamma gv\,\mathrm{d}x \in V^* \qquad \forall\, g \in L^2(\Gamma),$$

where V^* denotes the topological dual space of V.

Let $\mathcal{L}(t) : V \to V^*$ denote a second-order linear elliptic partial differential operator for $t \in [0,T]$. For example, $\mathcal{L}(t)$ could represent the weak form of the advection-reaction-diffusion operator

$$u \mapsto \{-\nabla\cdot(A(t,\cdot)\nabla u) + b(t,\cdot)\cdot\nabla u + c(t,\cdot)u\}, \tag{5.1}$$

where $A : \Omega_T \to \mathbb{R}^{d\times d}$ is the diffusivity tensor, $b : \Omega_T \to \mathbb{R}^d$ is an advection field, and $c : \Omega_T \to \mathbb{R}$ is a reaction coefficient. Here, ∇ refers to the derivative with respect to x. To guarantee the existence of solutions, we assume that the linear operator $\mathcal{L}$ is uniformly bounded and uniformly coercive, which we define below.

DEFINITION 5.1. *The operator $\mathcal{L}$ is uniformly bounded if for some $\varepsilon_0 > 0$ independent of $t \in [0,T]$,*

$$\langle \mathcal{L}(t)u, v\rangle_{V^*,V} \le \varepsilon_0 \|u\|_V \|v\|_V \quad \forall\, u,\, v \in V,$$

for almost all (a.a.) $t \in [0,T]$. Moreover, $\mathcal{L}$ is uniformly coercive if for some $\varepsilon_1 > 0$ independent of $t \in [0,T]$,

$$\varepsilon_1 \|v\|_V^2 \le \langle \mathcal{L}(t)v, v\rangle_{V^*,V} \quad \forall\, v \in V,$$

for a.a. $t \in [0,T]$. Here, $\langle \cdot,\cdot\rangle_{V^*,V}$ denotes the duality pairing between V^* and V.

In addition to being uniformly bounded and coercive, we assume that $t \mapsto \langle \mathcal{L}(t)u, v\rangle_{V^*,V}$ is measurable for all $u, v \in V$, $\beta \in L^2(0,T;V^*)$ is a forcing term, and $u_0 \in L^2(\Omega)$ is the initial state. We consider the optimal control problem

$$\text{(5.2a)} \qquad \text{minimize } \left\{ \frac{1}{2}\int_0^T \int_\Omega (u-w)^2 \,\mathrm{d}x\mathrm{d}t + \frac{\alpha}{2}\int_0^T \int_\Gamma z^2 \,\mathrm{d}x\mathrm{d}t \right\}$$

$$\text{subject to} \qquad u \in W(0,T),\;\; z \in L^2(\Gamma_T),$$

$$\text{(5.2b)} \qquad \begin{cases} \displaystyle\int_\Omega \frac{\partial u}{\partial t}(t,\cdot) v \,\mathrm{d}x + \langle [\mathcal{L}(t)u](t,\cdot), v\rangle_{V^*,V} & \\ \displaystyle\qquad\qquad = \langle \beta(t), v\rangle_{V^*,V} + \int_\Gamma z(t,\cdot) v \,\mathrm{d}x & \text{a.a. } t \in [0,T]\ \forall\, v \in V, \\ u(0,x) = u_0(x) & \text{a.a. } x \in \Omega, \end{cases}$$

where $\alpha > 0$ is the control penalty parameter, $w \in L^2(\Omega_T)$ is the desired state, and

$$W(0,T) := \{ v : \Omega_T \to \mathbb{R} \mid v \in L^2(0,T;V) \ \text{ and } \ \partial v/\partial t \in L^2(0,T;V^*) \}$$

is the solution space for (5.2b). In fact, under the stated assumptions, (5.2b) has a unique solution in $W(0,T)$ for any $z \in L^2(\Gamma_T)$ (cf. [20, Thm. 1.35]).

5.1. Discretization. To obtain a finite-dimensional approximation of (5.2), we discretize the PDE (5.2b) using Galerkin finite elements in space and implicit Euler in time. The subsequent results also hold for other time discretizations, including Crank–Nicolson or explicit Euler. We partition the time interval $[0,T]$ into N subintervals (t_{n-1}, t_n) with $0 = t_0 < t_1 < \cdots < t_{N-1} < t_N = T$ and denote the finite-element approximation space for the state by $V_M \subset V$, where M is the dimension of V_M. We further denote by $Z_m \subset L^2(\Gamma)$ the control approximation space where m is the dimension of Z_m. Using these spaces, we can write the discretized state equation as follows: for fixed $z_{m,n} \in Z_m$ with $n = 1, \ldots, N$, find $u_{M,n} \in V_M$ with $n = 1, \ldots, N$ such that $u_{M,0} = \widetilde{u}_0$, where $\widetilde{u}_0 \in V_M$ is an approximation of u_0 and

$$\begin{aligned} &\int_\Omega u_{M,n} v \,\mathrm{d}x + \delta t_n \langle \mathcal{L}(t_n) u_{M,n}, v\rangle_{V^*,V} \\ &= \int_\Omega u_{M,n-1} v \,\mathrm{d}x + \delta t_n \langle \beta(t_n), v\rangle_{V^*,V} + \delta t_n \int_\Gamma z_{m,n} v \,\mathrm{d}x \quad \forall\, v \in V_M, \end{aligned}$$

where $\delta t_n := t_n - t_{n-1}$. Given bases $\{\phi_i\}_{i=1}^M$ and $\{\psi_i\}_{i=1}^m$ of V_M and Z_m, respectively, we can rewrite the discretized PDE as the following linear system of equations: given $\mathbf{z}_n \in \mathbb{R}^m$ for $n = 1, \ldots, N$, find $\mathbf{u}_n \in \mathbb{R}^M$ such that

$$\text{(5.3)} \qquad c_n(\mathbf{u}_{n-1}, \mathbf{u}_n, \mathbf{z}_n) = (\mathbf{M} + \delta t_n \mathbf{K}_n)\mathbf{u}_n - \mathbf{M}\mathbf{u}_{n-1} - \delta t_n \mathbf{b}_n - \delta t_n \mathbf{B}\mathbf{z}_n = 0$$

for $n = 1, \dots, N$, where

$$[\mathbf{M}]_{ij} := \int_\Omega \phi_j \phi_i \,\mathrm{d}x, \quad [\mathbf{K}_n]_{ij} := \langle \mathcal{L}(t_n)\phi_j, \phi_i \rangle_{V^*,V},$$
$$[\mathbf{B}]_{ij} := \int_\Gamma \psi_j \phi_i \,\mathrm{d}x, \quad [\mathbf{b}_n]_j := \langle \beta(t_n), \phi_j \rangle_{V^*,V}.$$

With this notation, the discretized version of (5.2a) is

$$\begin{aligned} &\underset{\mathbf{U}\in\mathfrak{U},\,\mathbf{Z}\in\mathfrak{Z}}{\text{minimize}} \ \frac{1}{2}\sum_{n=1}^{N} \delta t_n \left\{ (\mathbf{u}_n - \mathbf{w}_n)^\top \mathbf{M}(\mathbf{u}_n - \mathbf{w}_n) + \alpha \mathbf{z}_n^\top \mathbf{R}\mathbf{z}_n \right\} \\ &\text{subject to} \quad (5.3), \end{aligned} \tag{5.4}$$

where we have approximated the temporal integral in the objective function using the right endpoint rule, $\mathbf{w}_n \in \mathbb{R}^M$ are the coefficients associated with an approximation of $w(t_n, \cdot)$ in V_M, and

$$[\mathbf{R}]_{ij} = \int_\Gamma \psi_j \psi_i \,\mathrm{d}x.$$

The assumptions on $\mathcal{L}$ and the choice of discretization ensure that $(\mathbf{M} + \delta t_n \mathbf{K}_n)$ is invertible for all $n = 1, \dots, N$ and therefore condition 2 in Assumption 1 is satisfied. In addition, since the dynamic constraint in (5.4) is linear in the state and control variables, and the objective function in (5.4) is quadratic, condition 3 in Assumption 1 is satisfied. Finally, since the matrices $(\mathbf{M} + \delta t_n \mathbf{K}_n)$ are invertible and the constraint is linear, the dynamic constraint has a unique solution that depends linearly on the control $\mathbf{Z} \in \mathfrak{Z}$. Therefore, condition 1 in Assumption 1 holds for any bounded set of controls. To verify (4.9), we employ stability estimates for (5.3).

5.2. Stability estimates. The linearity of (5.2b) and the uniform coercivity of $\mathcal{L}$ provide numerous convenient properties associated with the discretized PDE (5.3). In this section, we use these properties to ensure that the required assumptions for our sketching algorithm are satisfied. We first have the following error bound associated with the discretized state equation (5.3).

THEOREM 5.2. *Let $\bar{\mathbf{U}} \in \mathfrak{U}$ denote the solution to (5.3) for fixed control $\mathbf{Z} \in \mathfrak{Z}$, and let $\mathbf{U} \in \mathfrak{U}$ be arbitrary. Then the following bound holds:*

$$\|\mathbf{u}_n - \bar{\mathbf{u}}_n\|_{\mathbf{M}} \le (1 + \delta t_n \omega \varepsilon_1)^{-1} \left\{ \|\mathbf{u}_0 - \bar{\mathbf{u}}_0\|_{\mathbf{M}} + \sum_{i=1}^{n} \|c_i(\mathbf{u}_{i-1}, \mathbf{u}_i, \mathbf{z}_i)\|_{\mathbf{M}^{-1}} \right\}, \quad n = 1, \dots, N,$$

where $\bar{\mathbf{u}}_n$ and $\mathbf{u}_n$ are the nth subvectors in $\bar{\mathbf{U}}$ and $\mathbf{U}$, respectively, and $\omega > 0$ is the embedding constant associated with $V \hookrightarrow L^2(\Omega)$.

Proof. First, we write $c_n(\mathbf{u}_{n-1}, \mathbf{u}_n, \mathbf{z}_n)^\top (\mathbf{u}_n - \bar{\mathbf{u}}_n)$ in the form of (5.3). To this end, let $\bar{u}_{M,n}$, $u_{M,n} \in V_M$ denote the functions

$$\bar{u}_{M,n} = \sum_{i=1}^{M} [\bar{\mathbf{u}}_n]_i \phi_i \quad \text{and} \quad u_{M,n} = \sum_{i=1}^{M} [\mathbf{u}_n]_i \phi_i,$$

respectively, and let $\eta_n = (u_{M,n} - \bar{u}_{M,n})$. Then $c_n(\mathbf{u}_{n-1}, \mathbf{u}_n, \mathbf{z}_n)^\top (\mathbf{u}_n - \bar{\mathbf{u}}_n)$ is equal to

$$\begin{aligned}&\int_\Omega u_{M,n}\eta_n \,\mathrm{d}x + \delta t_n \langle \mathcal{L}(t_n)u_{M,n}, \eta_n\rangle_{V^*,V}\\ &- \int_\Omega u_{M,n-1}\eta_n \,\mathrm{d}x - \delta t_n\langle \beta(t_n), \eta_n\rangle_{V^*,V} - \delta t_n \int_\Gamma z_{m,n}\eta_n\,\mathrm{d}x\\ &= \int_\Omega \eta_n^2\,\mathrm{d}x + \delta t_n\langle \mathcal{L}(t_n)\eta_n, \eta_n\rangle_{V^*,V} - \int_\Omega \eta_{n-1}\eta_n\,\mathrm{d}x\end{aligned}$$

since $\bar{u}_{M,n}$ solves (5.3). The Cauchy–Schwarz inequality, the continuous embedding of V into $L^2(\Omega)$, and the uniform coercivity of $\mathcal{L}$ then ensure that

$$(1 + \delta t_n \omega\varepsilon_1)\|\mathbf{u}_n - \bar{\mathbf{u}}_n\|_{\mathbf{M}} \le \|c_n(\mathbf{u}_{n-1}, \mathbf{u}_n, \mathbf{z}_n)\|_{\mathbf{M}^{-1}} + \|\mathbf{u}_{n-1} - \bar{\mathbf{u}}_{n-1}\|_{\mathbf{M}}.$$

Repeated application of this inequality yields the desired bound. ∎

Theorem 5.2 ensures that the lower bound in (4.1) holds for all $\mathbf{Z} \in \mathfrak{Z}$ and $\mathbf{U} \in \mathfrak{U}$ (rather than only on some bounded sets $\mathfrak{Z}_0$ and $\mathfrak{U}_0$). The upper bound in (4.1) follows due to the linearity of c_n and holds again for all $\mathbf{Z} \in \mathfrak{Z}$ and $\mathbf{U} \in \mathfrak{U}$. Moreover, if $\mathbf{U}_r$ is the sketched state, then Theorem 5.2 yields

$$\|\mathbf{u}_{r,n} - \bar{\mathbf{u}}_n\|_{\mathbf{M}} \le (1 + \delta t_n\omega\varepsilon_1)^{-1}\sum_{i=1}^{n} \|c_i(\mathbf{u}_{r,i-1}, \mathbf{u}_{r,i}, \mathbf{z}_i)\|_{\mathbf{M}^{-1}}.$$

Next, we demonstrate that the adjoint error bound (4.2) holds globally as well. To this end, we write the adjoint equation associated with the discretized problem (5.4):

$$(\mathbf{M} + \delta t_N \mathbf{K}_N)^*\boldsymbol{\lambda}_N = -\delta t_N \mathbf{M}(\mathbf{u}_N - \mathbf{w}_N), \tag{5.5a}$$

$$(\mathbf{M} + \delta t_n \mathbf{K}_n)^*\boldsymbol{\lambda}_n = \mathbf{M}\boldsymbol{\lambda}_{n+1} - \delta t_n \mathbf{M}(\mathbf{u}_n - \mathbf{w}_n) \quad \text{for} \quad n = N-1, \ldots, 1. \tag{5.5b}$$

As before, we denote the solution to (5.5) with $\mathbf{u}_n$ replaced by $\bar{\mathbf{u}}_n$ for $n = 1, \ldots N$ by $\bar{\boldsymbol{\Lambda}}$. We have the following useful stability estimate associated with (5.5) that bounds the error in the adjoint.

Theorem 5.3. *Let $\bar{\boldsymbol{\Lambda}} \in \mathfrak{U}$ be the solution to the adjoint equation* (5.5) *associated with $\bar{\mathbf{U}} \in \mathfrak{U}$, and let $\boldsymbol{\Lambda}$, $\mathbf{U} \in \mathfrak{U}$ be arbitrary. Then the following bound holds:*

$$\begin{aligned}\|\boldsymbol{\lambda}_n - \bar{\boldsymbol{\lambda}}_n\|_{\mathbf{M}} \le (1 + \delta t_n\omega\varepsilon_1)^{-1}\Bigg\{&\sum_{j=n}^{N} \|(\mathbf{M} + \delta t_n \mathbf{K}_n)^*\boldsymbol{\lambda}_n - \mathbf{M}\boldsymbol{\lambda}_{n+1} + \delta t_n \mathbf{M}(\mathbf{u}_n - \mathbf{w}_n)\|_{\mathbf{M}^{-1}}\\ &+ \|\boldsymbol{\lambda}_N - \bar{\boldsymbol{\lambda}}_N\|_{\mathbf{M}} + \delta t_n \|\mathbf{u}_n - \bar{\mathbf{u}}_n\|_{\mathbf{M}}\Bigg\}, \quad n = 1, \ldots, N,\end{aligned} \tag{5.6}$$

where $\bar{\boldsymbol{\lambda}}_n$ and $\boldsymbol{\lambda}_n$ denote the nth subvectors of $\bar{\boldsymbol{\Lambda}}$ and $\boldsymbol{\Lambda}$, respectively, and $\omega > 0$ is the embedding constant associated with $V \hookrightarrow L^2(\Omega)$.

Proof. Using notation similar to that in the proof of Theorem 5.2, we can write the adjoint residual evaluated at $\boldsymbol{\Lambda}$ and $\mathbf{U}$ as

$$\begin{aligned}&\int_\Omega \lambda_{M,n} v\,\mathrm{d}x + \delta t_n \langle \mathcal{L}(t_n)^*\lambda_{M,n}, v\rangle_{V^*,V}\\ &\qquad - \int_\Omega \lambda_{M,n+1} v\,\mathrm{d}x + \delta t_n \int_\Omega (u_{M,n} - w_{M,n}) v\,\mathrm{d}x\end{aligned} \tag{5.7}$$

for $v \in V_M$, where $w_{M,n} \in V_M$ is the appropriate approximation of w. Evaluating this residual at $\bar{\mathbf{\Lambda}}$ and $\bar{\mathbf{U}}$ returns zero. Let $e_n = (\lambda_{M,n} - \bar{\lambda}_{M,n})$ and $\eta_n = (u_{M,n} - \bar{u}_{M,n})$. With this notation, (5.7) is equal to

$$\int_\Omega e_n v \,\mathrm{d}x + \delta t_n \langle \mathcal{L}(t_n)^* e_n, v\rangle_{V^*,V} - \int_\Omega e_{n+1} v \,\mathrm{d}x + \delta t_n \int_\Omega \eta_n v \,\mathrm{d}x.$$

Set $v = e_n$. Then applying the Cauchy–Schwarz inequality and using the uniform coercivity of $\mathcal{L}$ and the continuous embedding of V into $L^2(\Omega)$ yields

$$\begin{aligned}(1 + \delta t_n \omega \varepsilon_1)\|\boldsymbol{\lambda}_n - \bar{\boldsymbol{\lambda}}_n\|_{\mathbf{M}} \le \|(\mathbf{M} + \delta t_n \mathbf{K}_n)^* \boldsymbol{\lambda}_n - \mathbf{M}\boldsymbol{\lambda}_{n+1} + \delta t_n \mathbf{M}(\mathbf{u}_n - \mathbf{w}_n)\|_{\mathbf{M}^{-1}} \\ + \|\boldsymbol{\lambda}_{n+1} - \bar{\boldsymbol{\lambda}}_{n+1}\|_{\mathbf{M}} + \delta t_n \|\mathbf{u}_n - \bar{\mathbf{u}}_n\|_{\mathbf{M}}.\end{aligned}$$

Repeated application of this bound proves the desired result. ∎

By Theorems 5.2 and 5.3, we see that the adjoint error bound (4.2) holds globally. In particular, let $\mathbf{\Lambda}_r$ denote the solution to (5.5) associated with the sketched state $\mathbf{U}_r$. Then Theorems 5.2 and 5.3 ensure that

$$\begin{aligned}\|\boldsymbol{\lambda}_{r,n} - \bar{\boldsymbol{\lambda}}_n\|_{\mathbf{M}} &\le (1 + \delta t_n \omega \varepsilon_1)^{-1} \delta t_n \|\mathbf{u}_{r,n} - \bar{\mathbf{u}}_n\|_{\mathbf{M}} \\ &\le (1 + \delta t_n \omega \varepsilon_1)^{-2} \delta t_n \sum_{i=1}^{n} \|c_i(\mathbf{u}_{r,i-1}, \mathbf{u}_{r,i}, \mathbf{z}_i)\|_{\mathbf{M}^{-1}}.\end{aligned}$$

Hence Algorithm 4.4 ensures that the inexact gradient condition (4.9) is satisfied.

6. Numerical examples.

We demonstrate the effectiveness of the sketched trust-region algorithm on two PDE-constrained optimization problems. We present one example, the optimal control of an advection-reaction-diffusion equation, that satisfies the assumptions of the previous section and therefore is guaranteed to converge. We also present results on optimal flow control. This application is governed by the Navier–Stokes equations for which it is difficult to verify the assumptions of our theory, and so our algorithm does not necessarily admit guarantees for this problem. Nevertheless, we show remarkably good performance for this application.

In the numerics, we compute the function `ResidualNorm` using a domain-specific weighted norm (instead of the Frobenius norm) that respects the natural problem scaling. The guarantees of the method still hold: since all norms are equivalent in finite-dimensional vector spaces, we can ensure that the gradient error bound (4.9) holds (with a different value of the parameter θ) using any norm to measure the state residual. In addition, for both examples we set $\kappa_{\text{grad}} = 1$, $\Delta^{(0)} = 10$, $\eta_1 = 0.05$, $\eta_2 = 0.9$, and $\gamma = 0.25$ in Algorithm 4.5. We terminate the algorithm if the norm of the gradient is smaller than a prescribed tolerance `gtol` or when it exceeds a set maximum number of iterations `maxit`.

We note that the main computational cost of our approach is computing the objective function value, its gradient, and applying its Hessian to a vector. Sketching the various quantities is negligible in comparison. With this in mind, we only list the number of optimization iterations of each method for the following examples, as wall-clock time is proportional to the number of iterations. Additionally, in contrast to ROM techniques, there are no offline costs associated with our sketching approach.

6.1. Optimal control of an advection-reaction-diffusion equation.

For this example, our goal is to control the linear parabolic PDE (5.2b), where $\Omega = (0, 0.6) \times (0, 0.2)$, $\Gamma = \Omega$, and $\mathcal{L}$ is given by (5.1) with time-independent coefficients.

In particular, the forcing term β is the characteristic function of the intersection of the ball of radius 0.07 centered at $(0.1, 0.1)^\top$ with Ω the diffusivity coefficient $A = 0.1\,\mathbf{I}$, where $\mathbf{I} \in \mathbb{R}^{d\times d}$ is the identity matrix, the reaction coefficient is $c \equiv 1$, and the advection field is given by $b(x) = (7.5 - 2.5x_1, 2.5x_2)^\top$. We further supply (5.2b) with zero initial concentration $u_0 = 0$ and pure Neumann boundary conditions (i.e., $V = H^1(\Omega)$). Note that $\mathcal{L}$ is constant in time and is uniformly coercive since $\nabla \cdot b \equiv 0$ in Ω and $b \cdot n \geq 0$ on $\partial\Omega$, where n denotes the outward normal vector. Moreover, we set the target state $w \equiv 1$. Our optimization problem is then given by (5.2) with $\alpha = 10^{-4}$. We discretized (5.2b) in space using Q1 finite elements on a uniform mesh of 60×20 quadrilateral elements. In time, we discretize using implicit Euler with 500 equal time steps. This discretization results in $1{,}281 \times 500 = 640{,}500$ degrees of freedom. Moreover, the maximum possible rank of the state matrix is 500. Figure 1 depicts the tail energy and sketching error averaged over 20 realizations for the uncontrolled and optimal states. Both the tail energy and sketching error decay exponentially fast until saturating below $\mathcal{O}(10^{-12})$.

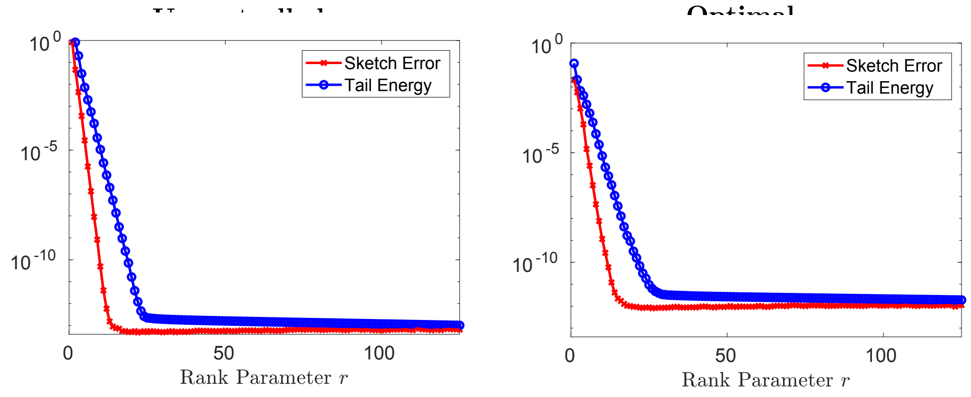

FIG. 1. *The sketching error averaged over* 20 *realizations and the tail energy for the uncontrolled state (left) and the optimal state (right) of the advection-reaction-diffusion example. Recall that the rank of the sketch is* $k = 2r + 1$.

We solved this problem using a Newton-based trust-region algorithm with fixed sketch rank and using Algorithm 4.5 with the rank update function $\mu(r, \tau) = \max\{r + 2, \lceil (p_2 - \log \tau)/p_1 \rceil\}$, where $p_1 > 0$ and $p_2 \in \mathbb{R}$ are computed by fitting a linear model of the logarithm of the average sketching error as a function of the rank for the uncontrolled state. For this problem, $p_1 = 2.6125$, $p_2 = 2.4841$, $\texttt{gtol} = 10^{-7}$, and $\texttt{maxit} = 20$. The final objective value, the iteration count, the number of function evaluations, the number of gradient evaluations, the cumulative truncated conjugate gradient (CG) iteration count, and the compression factor ζ are defined to be

$$\zeta := \frac{\text{full storage}}{\text{reduced storage}} = \frac{640{,}500}{k(1{,}281 + 500) + s^2},$$

where $k = 2r + 1$, $s = 2k + 1$ for each rank parameter r from 1 to 5 are displayed in Table 1. Notice that with rank 1 the algorithm did not converge, whereas the optimal objective function value is achieved up to 6 digits with rank 2. This is likely due to inaccuracies in the gradient. For this problem, the rank-2 sketch requires roughly three times more CG iterations (which dominate the computational work) than the full-storage algorithm; however, using the rank-2 sketch reduces the required memory by a factor of 70.96.

The iteration history of Algorithm 4.5 is listed in the top section of Table 2. For comparison, we have also listed the iteration history for the full-storage algorithm in the bottom section of Table 2. We notice that the sketched trust-region algorithm performs comparably to the full-storage algorithm but reduces the required memory by a factor of $\zeta = 23.14$ at the final iteration. It may be possible to further reduce the memory burden by tuning the parameter κ_{grad} or by employing a different rank update function μ. We further note that Algorithm 4.5 increases the rank three times (twice at iteration 2 and once at iteration 3), resulting in three additional state solves—a modest increase in computational work when compared to the reduction in required memory.

Table 1

Algorithmic performance summary for the advection-reaction-diffusion example for fixed rank, adaptive rank, and full storage: `objective` *is the final objective function value,* `iteration` *is the total number of iterations,* `nstate` *is the number of state solves,* `nadjoint` *is the number of adjoint solves,* `iterCG` *is the total number of truncated CG iterations, and* `compression` ζ *is the compression factor.* **The rank* 1 *experiment terminated because it exceeded the maximum number of iterations.*

rank	objective	iteration	nstate	nadjoint	iterCG	compression ζ
*1	5.544040e-4	20	21	11	196	118.79
2	5.528490e-4	5	6	6	151	70.96
3	5.528490e-4	4	5	5	78	50.46
4	5.528490e-4	4	5	5	67	38.08
5	5.528490e-4	4	5	5	59	31.83
Adaptive	5.528490e-4	4	8	5	65	23.14
Full	5.528490e-4	4	5	5	53	1.00

Table 2

The iteration histories for the adaptive rank (top) and full storage (bottom) algorithms: `iter` *is the iteration number,* `value` *is the objective function value,* `gnorm` *is the norm of the gradient,* `snorm` *is the norm of the step,* `delta` *is the trust-region radius,* `iterCG` *is the number of truncated CG iterations,* `rank` *is the rank of the sketch, and* `rnorm` *is the maximum residual norm associated with the sketched state. In the adaptive rank algorithm, the rank is updated using the rank update function* $\mu(r,\tau)$ *if the maximum residual norm exceeds the prescribed tolerance.*

	iter	value	gnorm	snorm	delta	iterCG	rank	rnorm
Adaptive	0	5.446e-2	5.990e-3	---	1.000e+1	---	1	2.707e-4
	1	1.375e-2	2.205e-3	1.000e+1	2.500e+1	1	1	1.990e-4
	2	1.475e-3	1.408e-4	2.499e+1	6.250e+1	5	5	6.700e-7
	3	5.531e-4	6.431e-7	4.077e+1	1.563e+1	27	7	3.893e-9
	4	5.528e-4	5.039e-9	1.059e+0	3.906e+2	32	7	1.508e-9
Full	0	5.446e-2	5.989e-3	---	1.000e+1	---	---	---
	1	1.375e-2	2.201e-3	1.000e+1	2.500e+1	1	---	---
	2	1.472e-3	1.401e-4	2.500e+1	6.250e+1	5	---	---
	3	5.538e-4	1.361e-6	4.051e+1	1.563e+1	19	---	---
	4	5.528e-4	8.416e-9	2.178e+0	3.906e+2	28	---	---

6.2. Optimal control of flow past a cylinder. For this example, we follow the problem set up in [18] and consider fluid flow past a cylinder. The cylinder impedes the flow; our goal is to rotate the cylinder to improve the flow rate. Formally, we let cylinder $C \subset \mathbb{R}^2$ denote the closed ball of radius $R = 0.5$ centered at the origin $x_0 = (0,0)^\top$, define the domain $D = (-15,45) \times (-15,15)$, and let $\Gamma_{\text{out}} = \{45\} \times (-15,15)$ denote the outflow boundary. We consider the optimal flow control problem

$$\underset{z\in L^2(0,T)}{\text{minimize}} \int_0^T \left\{ \int_{\partial C} \left(\frac{1}{\text{Re}} \frac{\partial \mathbf{v}}{\partial n} - pn \right) \cdot (z\tau - \mathbf{v}_\infty)\,\mathrm{d}x + \frac{\alpha}{2} z(t)^2 \right\} \mathrm{d}t, \quad \alpha > 0, \tag{6.1}$$

where the velocity and pressure pair $(\mathbf{v}, p) : [0, T] \times D \to \mathbb{R}^2 \times \mathbb{R}$ solves the Navier–Stokes equations

$$\frac{\partial \mathbf{v}}{\partial t} - \frac{1}{\mathrm{Re}}\Delta \mathbf{v} + (\mathbf{v} \cdot \nabla)\mathbf{v} + \nabla p = 0 \qquad \text{in } (0, T) \times D \setminus C, \tag{6.2a}$$

$$\nabla \cdot \mathbf{v} = 0 \qquad \text{in } (0, T) \times D \setminus C, \tag{6.2b}$$

$$\frac{1}{\mathrm{Re}}\frac{\partial \mathbf{v}}{\partial n} - pn = 0 \qquad \text{on } (0, T) \times \Gamma_{\mathrm{out}}, \tag{6.2c}$$

$$\mathbf{v} = \mathbf{v}_\infty \qquad \text{on } (0, T) \times \partial D \setminus \Gamma_{\mathrm{out}}, \tag{6.2d}$$

$$\mathbf{v} = z\tau \qquad \text{on } (0, T) \times \partial C \tag{6.2e}$$

with appropriately specified initial conditions on $\mathbf{v}$ and p. In (6.2), n is the outward normal vector on Γ_{out}, τ is the tangent vector

$$\tau(x) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} (x - x_0), \quad x \in \partial C,$$

and Re is the Reynold's number. Problem (6.1) minimizes the power required to overcome the drag on C. See [18, p. 87] for a comprehensive physical interpretation of this problem. The control action z defined in (6.2e) is the angular velocity of the cylinder. The vector $\mathbf{v}_\infty := (1, 0)^\top$ is the freestream velocity profile, and the boundary condition (6.2c) is stress-free. Similar to [18], we generate initial conditions for (6.2) by simulating (6.2) on the time interval $(-T_0, 0)$ for some $T_0 > 0$ starting with $\mathbf{v}(-T_0, \cdot)$ and $p(-T_0, \cdot)$ set to the potential flow around C [4]. The first row of Figure 2 depicts the computed initial velocity $\mathbf{v}_0$.

We discretized (6.2) in time with implicit Euler and approximated the temporal integral in (6.1) with the right end-point rule. Moreover, we discretized (6.2) in space using Q2–Q1 finite elements on the quadrilateral mesh depicted in Figure 3. The mesh contains 2,672 elements and 2,762 vertices. For our results, we set the time steps $\delta t_n = 0.025$, $T_0 = 80$, $T = 20$, and $\mathrm{Re} = 200$.

We refer the reader to [16, 18, 20] and the references therein for partial verification of Assumption 1 for various flow control problems.

The second row of Figure 2 depicts the optimal vorticity at the final time $t = 20$ (left) and the velocity field near the cylinder (right), while in the final row of Figure 2, we plot the computed optimal control. As seen in Figure 2, the optimal control effectively eliminates the vortex shedding seen in the first row of Figure 2 for the initial velocity. In Figure 4, we plot the sketching error averaged over 20 realizations and the tail energy (ranks 1 through 200) for the uncontrolled state (left) and the optimal state (right). We see that the decay in the sketching error and tail energy is roughly exponential, suggesting that our method should only require modest storage.

We solved the discretized optimization problem using a Newton-based trust-region algorithm with fixed sketch ranks $\{8, 16, 32, 64\}$ and using Algorithm 4.5 with the rank update function $\mu(r, \tau) = 2r$. The performance of the fixed-rank, adaptive-rank, and full storage experiments is summarized in Table 3. For each experiment, we set `gtol` $= 10^{-5}$ and `maxit` $= 40$. The only fixed-rank experiment to converge was rank 64. However, the rank-32 experiment produced an objective function value that was within 6 digits of the optimal value. The rank-32 experiment likely did not converge due to inaccuracies in the gradient. For the adaptive algorithm, we started with the initial rank set to 8. The behavior of the rank updates as well as the required gradient inexactness tolerances, and computed residual norms are plotted in

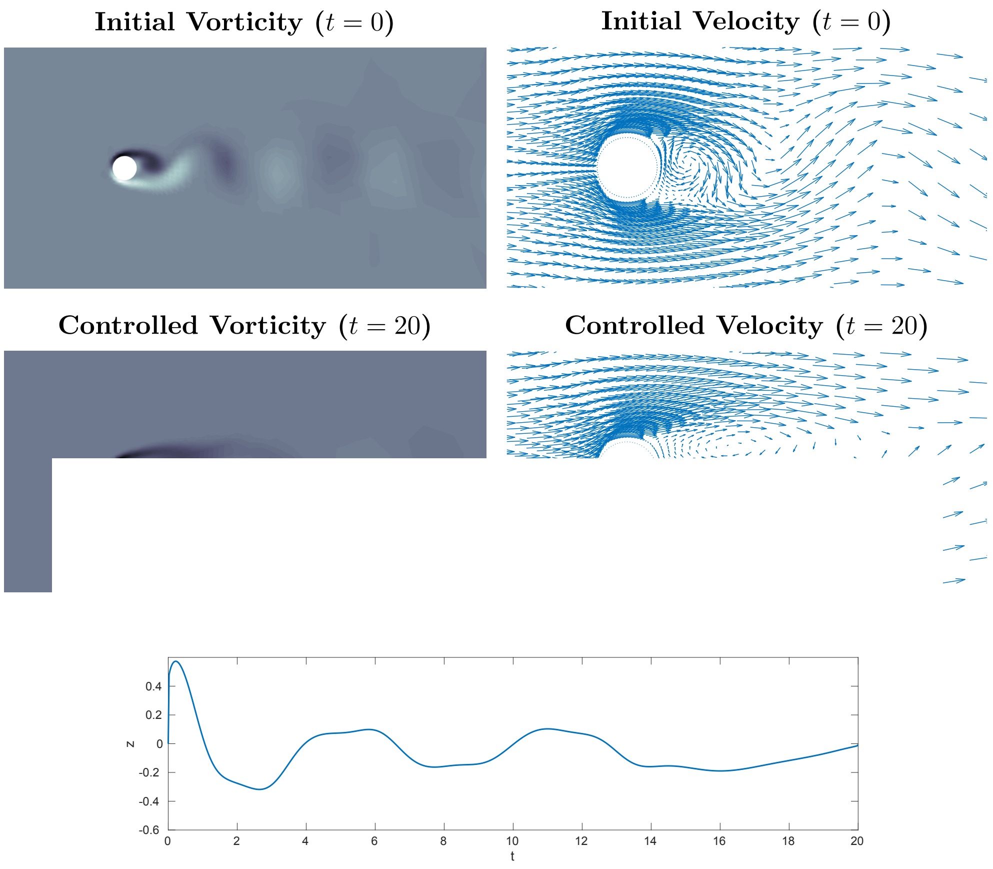

FIG. 2. *The initial velocity (first row) and the final, controlled velocity (second row). These images include the magnitude of vorticity* $(\nabla \times v)$ *on the subdomain* $[-5, 15] \times [-5, 5]$ *(left) and velocity* v *on the subdomain* $[-2, 6] \times [-2, 2]$ *(right). The bottom row depicts the computed optimal control.*

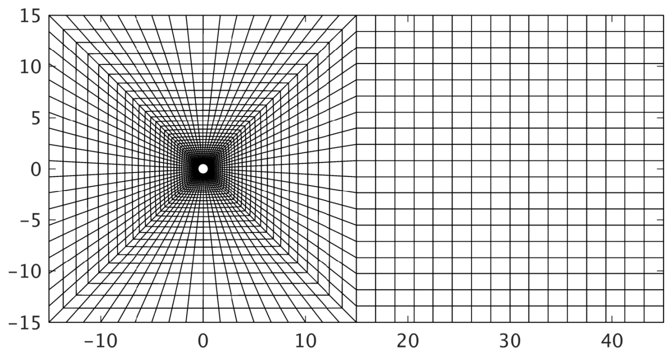

FIG. 3. *Quadrilateral mesh with* 2672 *elements and* 2762 *vertices.*

Figure 5. Algorithm 4.5 required comparable computation as the full-storage approach but reduced the memory by a factor of $\zeta = 5.88$ at the final iteration. The memory

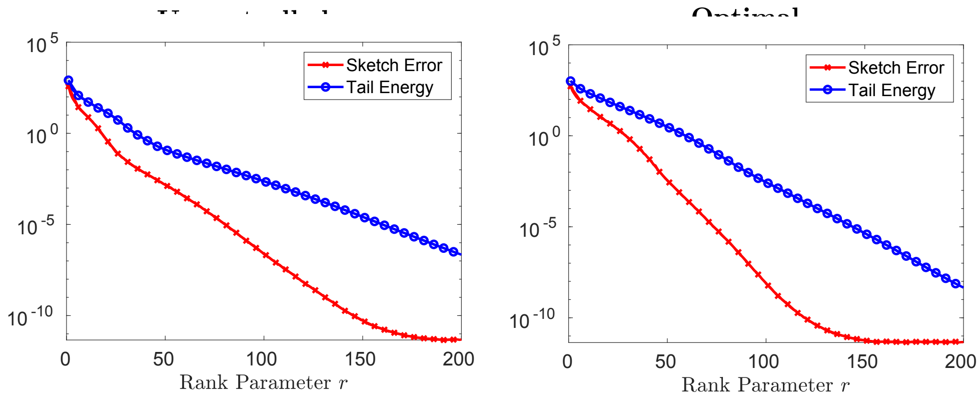

Fig. 4. *The sketching error averaged over* 20 *realizations and the tail energy for the uncontrolled state (left) and the optimal state (right) for the flow control example. Recall that the rank of the sketch is* $k = 2r + 1$.

burden could be further reduced by tuning κ_{grad} or by choosing a less aggressive rank update function μ. Similar to the example in section 6.1, the rank is increased three times (once at iterations 0, 1, and 10; see Figure 5), resulting in three additional state solves.

Table 3

Algorithmic performance summary for the flow control example for fixed rank, adaptive rank, and full storage: `objective` *is the final objective function value,* `iteration` *is the total number of iterations,* `nstate` *is the number of state solves,* `nadjoint` *is the number of adjoint solves,* `iterCG` *is the total number of truncated CG iterations, and* `compression` ζ *is the compression factor.* **The rank* 8, 16, *and* 32 *experiments terminated because they exceeded the maximum number of iterations.*

rank	objective	iteration	nstate	nadjoint	iterCG	compression ζ
*8	18.35919	40	41	15	136	45.44
*16	18.20003	40	41	33	897	23.35
*32	18.19779	40	41	31	236	11.80
64	18.19779	29	41	34	110	5.88
Adaptive	18.19779	23	27	24	121	5.88
Full	18.19779	29	30	24	107	---

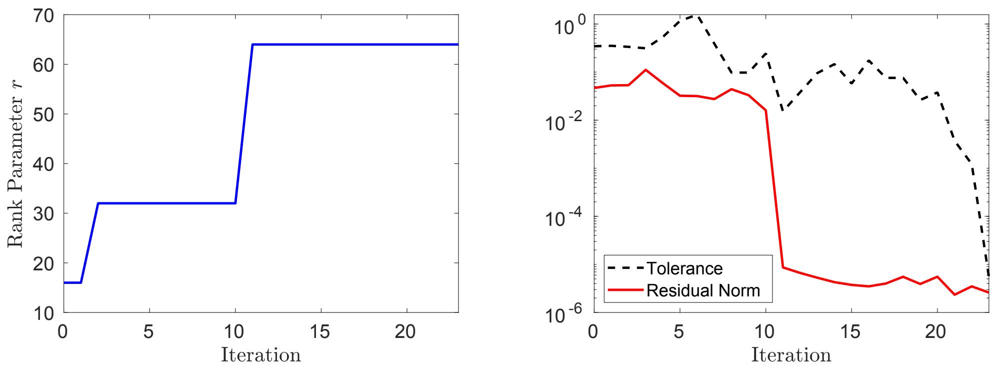

Fig. 5. *Inexact gradient behavior of Algorithm* 4.5 *applied to the flow control problem. Left: the sketch rank as a function of iteration. Right: the required gradient inexactness tolerance and computed residual norm as functions of iteration.*

Appendix A. Sketching routines. In this appendix, we provide pseudocode for the sketching algorithms described throughout the paper. In particular, we first present the abstract sketch class and then describe the methods required to apply the sketch-based approximation of the Hessian to a vector.

Algorithm A.1 Sketch class and methods.

```
1:  class SKETCH
2:      member variables $k$,$s$                                   sketch parameters
3:      member variables $\mathbf{\Upsilon}, \mathbf{\Omega}, \mathbf{\Phi}, \mathbf{\Psi}$                   random test matrices
4:      member variables $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$                       sketch matrices
5:      member variables $\mathbf{Q}, \mathbf{W}$                          low-rank factors
6:      member variables rec                             reconstruction flag
7:      function SKETCH($M$, $N$, rank $= r$)                 Sketch class constructor
8:          Initialize!($M$, $N$, rank $= r$)
9:          New(rec, $k, s, \mathbf{\Upsilon}, \mathbf{\Omega}, \mathbf{\Phi}, \mathbf{\Psi}, \mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{Q}, \mathbf{W}$)
10:     function INITIALIZE!($M$, $N$, rank $= r$)            Sketch class initializer
11:         rec $\leftarrow$ FALSE.
12:         $k \leftarrow 2r + 1$, $s \leftarrow 2k + 1$
13:         $\mathbf{\Upsilon} \leftarrow$ randn($k, M$), $\mathbf{\Omega} \leftarrow$ randn($k, N$)        Test matrix for range and co-range
14:         $\mathbf{\Phi} \leftarrow$ randn($s, M$) ,$\mathbf{\Psi} \leftarrow$ randn($s, N$)        Test matrices for core
15:         $\mathbf{X} \leftarrow$ zeros($k, N$), $\mathbf{Y} \leftarrow$ zeros($M, k$)        Approximation sketch of zero matrix
16:         $\mathbf{Z} \leftarrow$ zeros($s, s$)
17:         $\mathbf{Q} \leftarrow$ zeros($M, k$), $\mathbf{W} \leftarrow$ zeros($k, N$)        Low-rank factors
18:     function COLUMNUPDATE!($\mathbf{h}$, $n$, $\theta = 1.0$, $\eta = 1.0$) Update sketch matrices
19:     $\mathbf{X} \leftarrow \theta\mathbf{X} + \eta(\mathbf{\Upsilon}\mathbf{h})\mathbf{e}_n^*$
20:     $\mathbf{Y} \leftarrow \theta\mathbf{Y} + \eta\mathbf{h}(\mathbf{\Omega}\mathbf{e}_n)^*$
21:     $\mathbf{Z} \leftarrow \theta\mathbf{Z} + \eta(\mathbf{\Phi}\mathbf{h})(\mathbf{\Psi}\mathbf{e}_n)^*$
22: function RECONSTRUCT!( )                                  Reconstruct low-rank factors
23:     ($\mathbf{Q}$,$\mathbf{R}_2$) $\leftarrow$ qr($\mathbf{Y}, 0$)
24:     ($\mathbf{P}$,$\mathbf{R}_1$) $\leftarrow$ qr($\mathbf{X}^*, 0$)
25:     $\mathbf{C} \leftarrow ((\mathbf{\Phi}\mathbf{Q})\backslash\mathbf{Z})/((\Psi\mathbf{P})^*)$
26:     $\mathbf{W} \leftarrow \mathbf{C}\mathbf{P}^*$
27:     rec $\leftarrow$ TRUE
28: function COLUMN(j)                                       Reconstruct a single column
29:     if rec then
30:         return $\mathbf{Q} \cdot \mathbf{W}[:, j]$
```

Algorithm A.2 Compute residual norm for control $\mathbf{Z}$.

```
Input: A control iterate $\mathbf{Z} \in \mathbb{R}^{m\times N}$ and sketch object $\{\mathbf{U}\}_r$ for state $\mathbf{U} \in \mathbb{R}^{MN\times 1}$
Output: Residual norm: rnorm $= \|c(\widehat{\mathbf{U}}_r, \mathbf{Z})\|_2^2$
Storage: $\mathcal{O}(r(M+N))$
1: function ResidualNorm($\{\mathbf{U}\}_r$, $\mathbf{Z}$)
2:     $(\widehat{\mathbf{u}}_{\mathrm{curr}}, \mathrm{rnorm}) \leftarrow (\{\mathbf{U}\}_r.\mathtt{Column}(N), 0)$
3:     for $n \leftarrow N$ to 1 do
4:         $\widehat{\mathbf{u}}_{\mathrm{prev}} \leftarrow \{\mathbf{U}\}_r.\mathtt{Column}(n-1)$
5:         $\mathrm{rnorm} \leftarrow \mathrm{rnorm} + \|c_n(\widehat{\mathbf{u}}_{\mathrm{prev}}, \widehat{\mathbf{u}}_{\mathrm{curr}}, \mathbf{z}_n)\|_2^2$
6:         $\widehat{\mathbf{u}}_{\mathrm{curr}} \leftarrow \widehat{\mathbf{u}}_{\mathrm{prev}}$
7:     return rnorm
```

Algorithm A.3 Solve adjoint equation.

```
Input: Control $\mathbf{Z} \in \mathbb{R}^{m\times N}$ and sketch objects:
       $\{\mathbf{U}\}_r$ for state $\mathbf{U} \in \mathbb{R}^{M\times N}$
       $\{\mathbf{\Lambda}\}_r$ for adjoint $\mathbf{\Lambda} \in \mathbb{R}^{M\times N}$
Output: Updated adjoint sketch object $\{\mathbf{\Lambda}\}_r$
Storage: $\mathcal{O}((r_1+r_2)(M+N))$ for adjoint rank parameter $r_2 \le \min\{M,N\}$
1: function SolveAdjoint!($\{\mathbf{\Lambda}\}_r$, $\{\mathbf{U}\}_r$, $\mathbf{Z}$)
2:     $(\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{next}}) \leftarrow (\{\mathbf{U}\}_r.\mathtt{Column}(N-1), \{\mathbf{U}\}_r.\mathtt{Column}(N))$
3:     Solve the adjoint equation at index $N$ for $\widehat{\boldsymbol{\lambda}}_{\mathrm{next}}$,
           $$\mathsf{d}_2 c_N(\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{next}}, \mathbf{z}_N)\widehat{\boldsymbol{\lambda}}_{\mathrm{next}} = \mathsf{d}_2 f_N(\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{next}}, \mathbf{z}_N)$$
4:     $\{\mathbf{\Lambda}\}_r.\mathtt{ColumnUpdate!}(\widehat{\boldsymbol{\lambda}}_{\mathrm{next}}, N)$
5:     for $n = N-1$ to 1 do
6:         if $n = 1$ then
7:             $\widehat{\mathbf{u}}_{\mathrm{prev}} \leftarrow \mathbf{u}_0$
8:         else
9:             $\widehat{\mathbf{u}}_{\mathrm{prev}} \leftarrow \{\mathbf{U}\}_r.\mathtt{Column}(n-1)$
10:        Solve the adjoint equation at index $n$ for $\widehat{\boldsymbol{\lambda}}_{\mathrm{curr}}$,
           $$\mathsf{d}_2 c_n(\widehat{\mathbf{u}}_{\mathrm{prev}}, \widehat{\mathbf{u}}_{\mathrm{curr}}, \mathbf{z}_n)\widehat{\boldsymbol{\lambda}}_{\mathrm{curr}} = \mathsf{d}_2 f_n(\widehat{\mathbf{u}}_{\mathrm{prev}}, \widehat{\mathbf{u}}_{\mathrm{curr}}, \mathbf{z}_n) + \mathsf{d}_1 f_{n+1}(\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{next}}, \mathbf{z}_{n+1}) - \mathsf{d}_1 c_{n+1}(\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{next}}, \mathbf{z}_{n+1})\widehat{\boldsymbol{\lambda}}_{\mathrm{next}}$$
11:        $\{\mathbf{\Lambda}\}_r.\mathtt{ColumnUpdate!}(\widehat{\boldsymbol{\lambda}}_{\mathrm{curr}}, n)$
12:        $(\widehat{\mathbf{u}}_{\mathrm{next}}, \widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\boldsymbol{\lambda}}_{\mathrm{next}}) \leftarrow (\widehat{\mathbf{u}}_{\mathrm{curr}}, \widehat{\mathbf{u}}_{\mathrm{prev}}, \widehat{\boldsymbol{\lambda}}_{\mathrm{curr}})$
```

Algorithm A.4 Solve state sensitivity equation.

Input: Control $\mathbf{Z} \in \mathbb{R}^{m\times N}$, direction vector $\mathbf{V} \in \mathbb{R}^{m\times N}$, and sketch objects:
$\{\mathbf{U}\}_r$ for state $\mathbf{U} \in \mathbb{R}^{M\times N}$
$\{\mathbf{W}\}_r$ for state sensitivity $\mathbf{W} \in \mathbb{R}^{M\times N}$
Output: Updated state sensitivity sketch object $\{\mathbf{W}\}_r$
Storage: $\mathcal{O}((r_1 + r_3)(M + N))$ for state sensitivity rank parameter $r_3 \le \min\{M, N\}$

```
1: function SolveStateSensitivity!($\{\mathbf{W}\}_r$,$\{\mathbf{U}\}_r$, $\mathbf{Z}$, $\mathbf{V}$)
2:     for $n = 1$ to $N$ do
3:         if $n = 1$ then
4:             $(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{w}}_{\text{prev}}) \leftarrow (\mathbf{u}_0, \{\mathbf{U}\}_r.\text{Column}(1), \mathbf{0})$
5:         else
6:             $(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{w}}_{\text{prev}}) \leftarrow (\widehat{\mathbf{u}}_{\text{curr}}, \{\mathbf{U}\}_r.\text{Column}(n), \widehat{\mathbf{w}}_{\text{curr}})$
7:         Solve the state sensitivity equation at index $n$ for $\widehat{\mathbf{w}}_{\text{curr}}$

   $\mathsf{d}_2 c_n(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \mathbf{z}_n)\widehat{\mathbf{w}}_{\text{curr}} = \mathsf{d}_3 c_n(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \mathbf{z}_n)\mathbf{v}_n - \mathsf{d}_1 c_n(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \mathbf{z}_n)\widehat{\mathbf{w}}_{\text{prev}}$

8:         $\{\mathbf{W}\}_r$.ColumnUpdate!($\widehat{\mathbf{w}}_{\text{curr}}, n$)
```

Algorithm A.5 Apply fixed-rank Hessian approximation to a vector.

Input: Control $\mathbf{Z} \in \mathbb{R}^{m\times N}$, sketch object for state $\{\mathbf{U}\}_r$, direction $\mathbf{V} \in \mathbb{R}^{m\times N}$,
and rank parameters $r_2, r_3 \le \min\{M, N\}$
Output: Application of approximate Hessian to vector $\mathbf{V}$, $\tilde{\mathbf{H}} \approx \nabla^2 f(\mathbf{Z})\mathbf{V}$
Storage: $\mathcal{O}((r_1 + r_2 + r_3)(M + N))$

```
1: function ApplyFixedRankHessian($\mathbf{Z}$ , $\{\mathbf{U}\}_r$, $\mathbf{V}$, $r_2$, $r_3$)
2:     $\{\mathbf{\Lambda}\}_r \leftarrow$ Sketch($M, N, \text{rank} = r_2$)                      Initialize adjoint sketch object
3:     SolveAdjoint!($\{\mathbf{\Lambda}\}_r, \{\mathbf{U}\}_r, \mathbf{Z}$)                        Solve adjoint equation
4:     $\{\mathbf{\Lambda}\}_r$.Reconstruct!()                                   Get low-rank factors for adjoint
5:     $\{\mathbf{W}\}_r \leftarrow$ Sketch($M, N, \text{rank} = r_3$)                      Initialize state sensitivity sketch object
6:     SolveStateSensitivity!($\{\mathbf{W}\}_r, \{\mathbf{U}\}_r, \mathbf{Z}, \mathbf{V}$)     Solve state sensitivity equation
7:     $\{\mathbf{W}\}_r$.Reconstruct!()                                   Get low-rank factors for state sensitivity
8:     $\tilde{\mathbf{H}} \leftarrow$ ApplyHessian($\{\mathbf{W}\}_r, \{\mathbf{\Lambda}\}_r, \{\mathbf{U}\}_r, \mathbf{Z}, \mathbf{V}$)   Apply Hessian to $\mathbf{V}$
9:     return $\tilde{\mathbf{H}}$
```

Algorithm A.6 Apply Hessian to a vector using sketching.

```
Input: Control $\mathbf{Z} \in \mathbb{R}^{m\times N}$, direction vector $\mathbf{V} \in \mathbb{R}^{m\times N}$, and sketch objects:
        $\{\mathbf{U}\}_r$ for state $\mathbf{U} \in \mathbb{R}^{M\times N}$
        $\{\boldsymbol{\Lambda}\}_r$ for adjoint $\boldsymbol{\Lambda} \in \mathbb{R}^{M\times N}$
        $\{\mathbf{W}\}_r$ for state sensitivity $\mathbf{W} \in \mathbb{R}^{M\times N}$
Output: Application of approximate Hessian to vector $\mathbf{V}$, $\mathbf{H} \approx \nabla^2 f(\mathbf{Z})\mathbf{V}$
Storage: $\mathcal{O}((r_1+r_2+r_3)(M+N))$
 1: function ApplyHessian($\{\mathbf{W}\}_r$,$\{\boldsymbol{\Lambda}\}_r$,$\{\mathbf{U}\}_r$, $\mathbf{Z}$, $\mathbf{V}$)
 2:     $(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \widehat{\boldsymbol{\lambda}}_{\text{next}}) \leftarrow (\{\mathbf{U}\}_r.\text{Column}(N-1), \{\mathbf{U}\}_r.\text{Column}(N), \{\boldsymbol{\Lambda}\}_r.\text{Column}(N))$
 3:     $(\widehat{\mathbf{w}}_{\text{curr}}, \widehat{\mathbf{w}}_{\text{next}}) \leftarrow (\{\mathbf{W}\}_r.\text{Column}(N-1), \{\mathbf{W}\}_r.\text{Column}(N))$
 4:     Solve the adjoint sensitivity equation at index $N$ for $\widehat{\mathbf{p}}_{\text{next}}$,
            $(\mathsf{d}_2 c_N(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \mathbf{z}_N))^* \widehat{\mathbf{p}}_{\text{next}} = -\mathsf{d}_{2,3} L_N(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \mathbf{z}_N, \widehat{\boldsymbol{\lambda}}_{\text{next}})\mathbf{v}_N$
            $+ \mathsf{d}_{2,2} L_N(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \mathbf{z}_N, \widehat{\boldsymbol{\lambda}}_{\text{next}})\widehat{\mathbf{w}}_{\text{next}} + \mathsf{d}_{2,1} L_N(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \mathbf{z}_N, \widehat{\boldsymbol{\lambda}}_{\text{next}})\widehat{\mathbf{w}}_{\text{curr}}$
 5:     Apply Hessian of Lagrangian at index $N$,
            $\mathbf{h}_N = \mathsf{d}_{3,3} L_N(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \mathbf{z}_n, \widehat{\boldsymbol{\lambda}}_{\text{next}})\mathbf{v}_n - \mathsf{d}_{3,1} L_N(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \mathbf{z}_n, \widehat{\boldsymbol{\lambda}}_{\text{next}})\widehat{\mathbf{w}}_{\text{curr}}$
            $- \mathsf{d}_{3,2} L_N(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \mathbf{z}_n, \widehat{\boldsymbol{\lambda}}_{\text{next}})\widehat{\mathbf{w}}_{\text{next}} + (\mathsf{d}_3 c_N(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \mathbf{z}_N))^* \widehat{\mathbf{p}}_{\text{next}}$
 6:     for $n = N-1$ to 1 do
 7:         Solve the adjoint sensitivity equation at index $n$ for $\widehat{\mathbf{p}}_{\text{curr}}$,
            $(\mathsf{d}_2 c_n(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \mathbf{z}_n))^* \widehat{\mathbf{p}}_{\text{curr}} = -(\mathsf{d}_1 c_{n+1}(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \mathbf{z}_n))^* \widehat{\mathbf{p}}_{\text{next}}$
            $- \mathsf{d}_{2,3} L_n(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \mathbf{z}_n, \widehat{\boldsymbol{\lambda}}_{\text{curr}})\mathbf{v}_n - \mathsf{d}_{1,3} L_{n+1}(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \mathbf{z}_n, \widehat{\boldsymbol{\lambda}}_{\text{next}})\mathbf{v}_{n+1}$
            $+ \mathsf{d}_{2,2} L_n(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \mathbf{z}_n, \widehat{\boldsymbol{\lambda}}_{\text{curr}})\widehat{\mathbf{w}}_{\text{curr}} + \mathsf{d}_{1,2} L_{n+1}(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \mathbf{z}_n, \widehat{\boldsymbol{\lambda}}_{\text{next}})\widehat{\mathbf{w}}_{\text{next}}$
            $+ \mathsf{d}_{2,1} L_n(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \mathbf{z}_n, \widehat{\boldsymbol{\lambda}}_{\text{curr}})\widehat{\mathbf{w}}_{\text{prev}} + \mathsf{d}_{1,1} L_{n+1}(\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{next}}, \mathbf{z}_n, \widehat{\boldsymbol{\lambda}}_{\text{next}})\widehat{\mathbf{w}}_{\text{curr}}.$
 8:         Apply Hessian of Lagrangian at index $n$
            $\mathbf{h}_n \leftarrow \mathsf{d}_{3,3} L_n(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \mathbf{z}_n, \widehat{\boldsymbol{\lambda}}_{\text{curr}})\mathbf{v}_n - \mathsf{d}_{3,1} L_n(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \mathbf{z}_n, \widehat{\boldsymbol{\lambda}}_{\text{curr}})\widehat{\mathbf{w}}_{\text{prev}}$
            $- \mathsf{d}_{3,2} L_n(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \mathbf{z}_n, \widehat{\boldsymbol{\lambda}}_{\text{curr}})\widehat{\mathbf{w}}_{\text{curr}} + (\mathsf{d}_3 c_n(\widehat{\mathbf{u}}_{\text{prev}}, \widehat{\mathbf{u}}_{\text{curr}}, \mathbf{z}_n))^* \widehat{\mathbf{p}}_{\text{curr}}$
 9:         $(\widehat{\mathbf{u}}_{\text{next}}, \widehat{\mathbf{u}}_{\text{curr}}, \widehat{\boldsymbol{\lambda}}_{\text{next}}) \leftarrow (\widehat{\mathbf{u}}_{\text{curr}}, \widehat{\mathbf{u}}_{\text{prev}}, \widehat{\boldsymbol{\lambda}}_{\text{curr}})$
10:         $(\widehat{\mathbf{w}}_{\text{next}}, \widehat{\mathbf{w}}_{\text{curr}}, \widehat{\mathbf{p}}_{\text{next}}) \leftarrow (\widehat{\mathbf{w}}_{\text{curr}}, \widehat{\mathbf{w}}_{\text{prev}}, \widehat{\mathbf{p}}_{\text{curr}})$
11:     return $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_N]$
```

Acknowledgments. We thank the editorial staff and anonymous reviewers for their helpful comments.

REFERENCES

[1] A. C. Antoulas, *Approximation of Large-Scale Dynamical Systems*, Adv. Des. Control 6, SIAM, Philadelphia, 2005, https://doi.org/10.1137/1.9780898718713.

[2] S. R. Arridge and J. C. Schotland, *Optical tomography: Forward and inverse problems*, Inverse Problems, 25 (2009),123010, https://doi.org/10.1088/0266-5611/25/12/123010.

[3] G. Aupy, J. Herrmann, P. Hovland, and Y. Robert, *Optimal multistage algorithm for adjoint computation*, SIAM J. Sci. Comput., 38 (2016), pp. C232–C255, https://doi.org/10.1137/15M1019222.

[4] G. Batchelor, *An Introduction to Fluid Dynamics*, Cambridge Math. Lib., Cambridge University Press, Cambridge, UK, 1999, https://doi.org/10.1017/CBO9780511800955.

[5] P. Benner, E. Sachs, and S. Volkwein, *Model order reduction for PDE constrained optimization*, in Trends in PDE Constrained Optimization, Birkhäuser/Springer, Cham, 2014, pp. 303–326.

[6] C. Boutsidis, D. P. Woodruff, and P. Zhong, *Optimal principal component analysis in distributed and streaming models*, in Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing, 2016, pp. 236–249, https://doi.org/10.1145/2897518.2897646.

[7] K. L. Clarkson and D. P. Woodruff, *Numerical linear algebra in the streaming model*, in Proceedings of the Forty-First ACM Symposium on Theory of Computing, 2009, https://doi.org/10.1145/1536414.1536445.

[8] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust Region Methods*, SIAM, Philadelphia, 2000, https://doi.org/10.1137/1.9780898719857.

[9] E. C. Cyr, J. N. Shadid, and T. Wildey, *Towards efficient backward-in-time adjoint computations using data compression techniques*, Comput. Methods Appl. Mech. Engrg., 288 (2015), pp. 24–44.

[10] L. Dedè, *Reduced basis method and a posteriori error estimation for parametrized linear-quadratic optimal control problems*, SIAM J. Sci. Comput., 32 (2010), pp. 997–1019, https://doi.org/10.1137/090760453.

[11] M. Fahl and E. Sachs, *Reduced order modelling approaches to PDE-constrained optimization based on proper orthogonal decomposition*, in Large-Scale PDE-Constrained Optimization, Lect. Notes Comput. Sci. Eng. 30, L. T. Biegler, O. Ghattas, M. Heinkenschloss, and B. van Bloemen Waanders, eds., Springer-Verlag, Berlin, 2003, https://doi.org/10.1007/978-3-642-55508-4_16.

[12] G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins Stud. Math. Sci., Johns Hopkins University Press, Baltimore, MD, 1996.

[13] S. Götschel, N. Chamakuri, K. Kunisch, and M. Weiser, *Lossy compression in optimal control of cardiac defibrillation*, J. Sci. Comput., 60 (2014), pp. 35–59.

[14] S. Götschel, C. von Tycowicz, K. Polthier, and M. Weiser, *Reducing memory requirements in scientific computing and optimal control*, in Multiple Shooting and Time Domain Decomposition Methods, Springer, Cham, 2015, pp. 263–287.

[15] A. Griewank and A. Walther, *Algorithm* 799*: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation*, ACM Trans. Math. Software, 26 (2000), pp. 19–45, https://doi.org/10.1145/347837.347846.

[16] M. D. Gunzburger, *Perspectives in Flow Control and Optimization*, SIAM, Philadelphia, 2002, https://doi.org/10.1137/1.9780898718720.

[17] N. Halko, P. G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288, https://doi.org/10.1137/090771806.

[18] J.-W. He, R. Glowinski, R. Metcalfe, A. Nordlander, and J. Periaux, *Active control and drag optimization for flow past a circular cylinder:* I*. Oscillatory cylinder rotation*, J. Comput. Phys., 163 (2000), pp. 83–117, https://doi.org/10.1006/jcph.2000.6556.

[19] M. Heinkenschloss and L. N. Vicente, *Analysis of inexact trust-region SQP algorithms*, SIAM J. Optim., 12 (2001), pp. 283–302, https://doi.org/10.1137/S1052623499361543.

[20] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich, *Optimization with PDE Constraints*, Math. Model. Theory Appl. 23, Springer, New York, 2009, https://doi.org/10.1007/978-1-4020-8839-1.

[21] A. A. Jalali, C. S. Sims, and P. Famouri, *Reduced Order Systems*, Lect. Notes Control Inf.

Sci. 343, Springer-Verlag, Berlin, 2006, https://doi.org/10.1007/11597018.

[22] C. Kaebe, J. H. Maruhn, and E. W. Sachs, *Adjoint-based Monte Carlo calibration of financial market models*, Finance Stoch., 13 (2009), pp. 351–379, https://doi.org/10.1007/s00780-009-0097-9.

[23] A. D. Klose and A. H. Hielscher, *Optical tomography using the time-independent equation of radiative transfer—Part 2: Inverse model*, J. Quant. Spectrosc. Radiat. Transf., 72 (2002), pp. 715–732, https://doi.org/10.1016/S0022-4073(01)00151-0.

[24] D. P. Kouri, M. Heinkenschloss, D. Ridzal, and B. G. van Bloemen Waanders, *A trust-region algorithm with adaptive stochastic collocation for PDE optimization under uncertainty*, SIAM J. Sci. Comput., 35 (2013), pp. A1847–A1879, https://doi.org/10.1137/120892362.

[25] D. P. Kouri and D. Ridzal, *Inexact trust-region methods for PDE-constrained optimization*, in Frontiers in PDE-Constrained Optimization, Springer, New York, 2018, pp. 83–121, https://doi.org/10.1007/978-1-4939-8636-1_3.

[26] J. R. Krebs, J. E. Anderson, D. Hinkley, R. Neelamani, S. Lee, A. Baumstein, and M.-D. Lacasse, *Fast full-wavefield seismic inversion using encoded sources*, Geophysics, 74 (2009), pp. WCC177–WCC188, https://doi.org/10.1190/1.3230502.

[27] M.-D. Lacasse, L. White, H. Denli, and L. Qiu, *Full-wavefield inversion: An extreme-scale PDE-constrained optimization Problem*, in Frontiers in PDE-Constrained Optimization, Springer New York, 2018, pp. 205–255, https://doi.org/10.1007/978-1-4939-8636-1_6.

[28] C. Lee, J. Kim, and H. Choi, *Suboptimal control of turbulent channel flow for drag reduction*, J. Fluid Mech., 358 (1998), p. 245–258, https://doi.org/10.1017/S002211209700815X.

[29] M. W. Mahoney, *Randomized algorithms for matrices and data*, Found. Trends Mach. Learn., 3 (2011), pp. 123–224, https://doi.org/10.1561/2200000035.

[30] P. Stumm and A. Walther, *New algorithms for optimal online checkpointing*, SIAM J. Sci. Comput., 32 (2010), pp. 836–854, https://doi.org/10.1137/080742439.

[31] Y. Sun, Y. Guo, C. Luo, J. A. Tropp, and M. Udell, *Low-rank Tucker approximation of a tensor from streaming data*, SIAM J. Math. Data Sci. 2 (2020), pp. 1123–1150, https://doi.org/10.1137/19M1257718.

[32] Y. Sun, Y. Guo, J. A. Tropp, and M. Udell, *Tensor random projection for low memory dimension reduction*, in NeurIPS Workshop on Relational Representation Learning, 2018.

[33] A. Tarantola, *Linearized inversion of seismic reflection data*, Geophys. Prospect., 32 (1984), pp. 998–1015, https://doi.org/10.1111/j.1365-2478.1984.tb00751.x.

[34] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Fixed-Rank Approximation of a Positive-Semidefinite Matrix from Streaming Data*, Adv. Neural Inform. Process. Syst. 30, 2017.

[35] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Practical sketching algorithms for low-rank matrix approximation*, SIAM J. Matrix Anal. Appl., 38 (2017), pp. 1454–1485, https://doi.org/10.1137/17M1111590.

[36] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, *Streaming low-rank matrix approximation with an application to scientific simulation*, SIAM J. Sci. Comput., 41 (2019), pp. A2430–A2463, https://doi.org/10.1137/18M1201068.

[37] Q. Wang, P. Moin, and G. Iaccarino, *Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation*, SIAM J. Sci. Comput., 31 (2009), pp. 2549–2567, https://doi.org/10.1137/080727890.

[38] M. Warner and L. Guasch, *Adaptive waveform inversion: Theory*, GEOPHYSICS, 81 (2016), pp. R429–R445, https://doi.org/10.1190/geo2015-0387.1.

[39] D. P. Woodruff, *Sketching as a tool for numerical linear algebra*, Found. Trends Theor. Comput. Sci., 10 (2014), pp. 1–157, https://doi.org/10.1561/0400000060.

[40] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert, *A fast randomized algorithm for the approximation of matrices*, Appl. Comput. Harmon. Anal., 25 (2008), pp. 335–366, https://doi.org/10.1016/j.acha.2007.12.002.

[41] M. J. Zahr, K. T. Carlberg, and D. P. Kouri, *An efficient, globally convergent method for optimization under uncertainty using adaptive model reduction and sparse grids*, SIAM/ASA J. Uncertain. Quantif., 7 (2019), pp. 877–912, https://doi.org/10.1137/18M1220996.