SAND2025-12569J

A Randomized Sketching Trust-Region Secant Method for Low-Memory Dynamic Optimization

Drew Kouri[1†] and Radoslav Vuchkov[2†]

[1]Optimization and Uncertainty Quantification, Sandia National Laboratories, P.O. Box 5800, Albuquerque, 87125, NM, USA.
[2]Computational Mathematics, Sandia National Laboratories, P.O. Box 5800, Albuquerque, 87125, NM, USA.

Contributing authors: dpkouri@sandia.gov; rgvuchk@sandia.gov;
[†]These authors contributed equally to this work.

Abstract

The numerical solution of dynamic optimization problems is often limited by the memory required to store the state trajectory, which is used to evaluate the objective function and its derivatives. Recently, [R. Muthukumar et al., *SIAM Journal on Optimization* 31(2), pp. 1242–1275 (2021)] introduced a trust-region method for dynamic optimization that employs randomized sketching to compress the state trajectory, resulting in inexact derivative computations. By adaptively learning the sketch rank, the trust-region algorithm achieves rigorous convergence guarantees. We extend this approach to use secant Hessian approximations. Due to the randomness introduced by the sketch, the traditional secant update formulae can produce poor Hessian approximations. In particular, the difference of two gradients, computed from two different sketches, may be inconsistent. To overcome this, we employ a sketched approximation of the Hessian application, in lieu of computing the gradient difference. We numerically demonstrate the improved stability of this approach on an example from PDE-constrained optimization.

Keywords: Quasi-Newton, Trust Regions, Randomized Sketching, Dynamic Optimization

MSC Classification: 49M15 , 49M37 , 65K05 , 65K10 , 90C06 , 90C30

1 Introduction

Large-scale dynamic optimization problems are ubiquitous in engineering and science applications such as full waveform inversion [1–3] and optical tomography [4, 5]. In these applications, the state trajectory is often too large to store in memory. To overcome this issue, the authors in [6], introduced a low-memory approach for solving the discrete-time dynamic optimization problem

$$\begin{aligned} \min_{u_n \in \mathbb{R}^M,\, z_n \in \mathbb{R}^m} & \sum_{n=1}^{N} f_n(u_{n-1}, u_n, z_n) \\ \text{subject to} \qquad & c_n(u_{n-1}, u_n, z_n) = 0, \quad n = 1, \ldots, N, \end{aligned} \tag{1}$$

where $z_n \in \mathbb{R}^m$ and $u_n \in \mathbb{R}^M$ are the control actions and system states at the nth time step, respectively, $u_0 \in \mathbb{R}^M$ 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 the cost 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 u_{n-1} into u_n.

Memory limits often constrain numerical algorithms for solving (1). For example, sequential quadratic programming methods require the storage of the entire state trajectory $\{u_n\}_{n=1}^N$, the Lagrange multipliers associated with the N constraints, and the controls $\{z_n\}_{n=1}^N$. In total, one must store at least $(2M+m)N$ floating point numbers. However, in real applications, M and N can be extremely large, e.g., $M \approx 6.4 \times 10^{10}$ and $N \approx 4 \times 10^5$ for subsurface full waveform inversion problems [1]. On the other hand, when the dynamic constraint in (1) is uniquely solvable, one can reformulate (1) as a reduced optimization problem by eliminating the state trajectory. The optimization variable in this reduced approach is only the control $\{z_n\}_{n=1}^N$, which consists of mN floating point numbers, significantly reducing the required storage. However, the computational cost is increased as each evaluation of the objective function requires solving the dynamic constraint. Worse, evaluating the gradient of the objective function requires solving the backward-in-time adjoint equation [7], which depends on the entire state trajectory.

Such requirements force algorithm developers to make hard choices—either to reduce the fidelity of the model using, e.g., reduced-order models (ROMs) or to repeat computation using checkpointing. While ROMs are an excellent solution in some cases, they are often tailored for specific models and demand significant domain expertise. Furthermore, ROMs can be incompatible with legacy codes as they often require invasive modification of the simulation software. However, ROM development is an active area of research and adaptive ROM construction can work well for solving certain classes of optimization problems [8, 9]. See [10] for an in-depth review of ROM techniques for PDE-constrained optimization. Checkpointing methods are a popular alternative, where the backward pass is preformed without storing the full state [2, 11–13]. These methods store judiciously chosen snapshots of the state variables u_n and then recompute the state from these checkpoints to solve the adjoint equation. This procedure results in lower memory requirements, but increases the cost of computing gradient and Hessian information. For example, if one can store at most k state

vectors, then the checkpointing strategy described in [12] will perform

$$w(N,k) := \tau N - \beta(k+1, \tau-1) \tag{2}$$

additional state time steps to compute the gradient, where

$$\beta(s,t) := \binom{s+t}{s}$$

and τ is the unique integer satisfying $\beta(k,\tau-1) < N \leq \beta(k,\tau)$. Note that the computation of higher-order derivatives compounds this cost, making application of the Hessian very costly.

As an alternative, randomized sketching [14–16] can be used to compress the state trajectory, reducing the memory to $\mathcal{O}(N+M)$, at the cost of inexact gradients. The trust-region algorithm introduced in [6] adaptively learns the sketch rank based on the algorithm progress, yielding rigorous convergence guarantees. In their numerical results, the authors employ a conjugate gradient method with approximate Hessian applications, where the auxiliary dynamical systems solutions (i.e., the adjoint and state sensitivity trajectories) are also sketched. In this paper, we investigate the effect of the randomized sketching on secant (e.g., BFGS and SR1) approximations of the Hessian. The randomness introduced by sketching can lead to inconsistencies between consecutive gradient approximations, producing poor curvature estimates. This has been observed in machine learning with stochastic secant methods [17]. Inspired by [17], we employ inexact sketched Hessian applications as a way to stabilize the curvature estimates. Using a numerical example from PDE-constrained optimization, we demonstrate that the sketched Hessian applications lead to superior performance.

2 Problem formulation

Throughout, $\mathcal{Z} := \mathbb{R}^{mN}$ denotes the control space and $\mathcal{U} := \mathbb{R}^{MN}$ denotes the state space. Moreover, we denote the stacked control and state vectors by

$$\boldsymbol{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_N \end{bmatrix} \in \mathcal{Z} \qquad \text{and} \qquad \boldsymbol{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix} \in \mathcal{U},$$

respectively. Using this notation, we rewrite the dynamic optimization problem (1) as

$$\min_{\boldsymbol{u}\in\mathcal{U},\, \boldsymbol{z}\in\mathcal{Z}} f(\boldsymbol{u},\boldsymbol{z}) \qquad \text{subject to} \qquad c(\boldsymbol{u},\boldsymbol{z}) = 0, \tag{3}$$

where $f : \mathcal{U}\times\mathcal{Z} \to \mathbb{R}$ and $c : \mathcal{U}\times\mathcal{Z} \to \mathcal{U}$ are defined by

$$f(\boldsymbol{u},\boldsymbol{z}) := \sum_{n=1}^{N} f_n(u_{n-1}, u_n, z_n) \qquad \text{and} \qquad c(\boldsymbol{u},\boldsymbol{z}) := \begin{bmatrix} c_1(u_0,u_1,z_1) \\ \vdots \\ c_N(u_{N-1},u_N,z_N) \end{bmatrix},$$

respectively. We assume that f and c are twice continuously differentiable on $\mathcal{U} \times Z$, that there exists a unique control-to-state map $S : \mathcal{U} \to \mathcal{Z}$, i.e., for any control $\boldsymbol{z} \in \mathcal{Z}$ there exists a unique trajectory $S(\boldsymbol{z}) \in \mathcal{U}$ that satisfies the dynamic constraint $c(S(\boldsymbol{z}), \boldsymbol{z}) = 0$, and that the state Jacobian $c_{\boldsymbol{u}}(S(\boldsymbol{z}), \boldsymbol{z})$ has a bounded inverse for all controls $\boldsymbol{z} \in \mathcal{Z}$. Under these assumptions, the implicit function theorem ensures that S is twice continuously differentiable. Giving the sequential nature of c, the state trajectory $\overline{\boldsymbol{u}} = S(\boldsymbol{z})$ can be expressed in terms of the time steps as

$$S(\boldsymbol{z}) = \begin{bmatrix} S_1(u_0, z_1) \\ S_2(S_1(u_0, z_1), z_2) \\ \vdots \\ S_N(S_{N-1}(\ldots, z_{N-1}), z_n) \end{bmatrix},$$

where $\overline{u}_n = S_n(\overline{u}_{n-1}, z_n) \in \mathbb{R}^M$ denotes the unique solution to the nth constraint

$$c_n(\overline{u}_{n-1}, \overline{u}_n, z_n) = 0 \qquad \text{for} \qquad n = 1, \ldots, N$$

Finally, employing the control-to-state map S, we can reformulate (3) as the reduced dynamic optimization problem

$$\min_{\boldsymbol{z} \in \mathcal{Z}} \{F(\boldsymbol{z}) := f(S(\boldsymbol{z}), \boldsymbol{z})\}. \tag{4}$$

As in [6], our method compresses the dynamic state $S(\boldsymbol{z})$ at each iteration for use in the gradient computation. Employing the adjoint method, we compute the gradient of F as

$$\nabla F(\boldsymbol{z}) = c_{\boldsymbol{z}}(S(\boldsymbol{z}), \boldsymbol{z})^\top \boldsymbol{\lambda} + f_{\boldsymbol{z}}(S(\boldsymbol{z}), \boldsymbol{z}), \tag{5}$$

where $\boldsymbol{\lambda} = \Lambda(\boldsymbol{z}) \in \mathcal{U}$ solves the adjoint equation

$$c_{\boldsymbol{u}}(S(\boldsymbol{z}), \boldsymbol{z})^\top \boldsymbol{\lambda} = -f_{\boldsymbol{u}}(S(\boldsymbol{z}), \boldsymbol{z}). \tag{6}$$

Recall that the adjoint equation is solved backward in time and at the nth time step, the adjoint variable $\lambda_n \in \mathbb{R}^M$ solves

$$\begin{cases} \frac{\partial c_N}{\partial u_N}(\overline{u}_{N-1}, \overline{u}_N, z_N)^\top \lambda_N = -\frac{\partial f_N}{\partial u_N}(\overline{u}_{N-1}, \overline{u}_N, z_N), \\ \frac{\partial c_n}{\partial u_n}(\overline{u}_{n-1}, \overline{u}_n, z_n)^\top \lambda_n = -\left[\frac{\partial f_n}{\partial u_n}(\overline{u}_{n-1}, \overline{u}_n, z_n) + \frac{\partial f_{n+1}}{\partial u_n}(\overline{u}_n, \overline{u}_{n+1}, z_{n+1})\right] \\ \qquad\qquad - \frac{\partial c_{n+1}}{\partial u_n}(\overline{u}_n, \overline{u}_{n+1}, z_{n+1})^\top \lambda_{n+1}, \quad n = N-1, \ldots, 1. \end{cases}$$

The application of the Hessian to a vector $\boldsymbol{v}$ is computed in a similar manner as the gradient and is given by

$$\nabla^2 F(\boldsymbol{z})\boldsymbol{v} = c_{\boldsymbol{z}}(S(\boldsymbol{z}), \boldsymbol{z})^\top \boldsymbol{\pi} + L_{\boldsymbol{zu}}(S(\boldsymbol{z}), \boldsymbol{z}, \Lambda(\boldsymbol{z}))\boldsymbol{w} + L_{\boldsymbol{zz}}(S(\boldsymbol{z}), \boldsymbol{z}, \Lambda(\boldsymbol{z}))\boldsymbol{v},$$

where $L : \mathcal{U} \times \mathcal{Z} \times \mathcal{U} \to \mathbb{R}$ is the Lagrangian functional

$$L(\boldsymbol{u}, \boldsymbol{z}, \boldsymbol{\lambda}) := f(\boldsymbol{u}, \boldsymbol{z}) + \boldsymbol{\lambda}^\top c(\boldsymbol{u}, \boldsymbol{z}),$$

$\boldsymbol{w} = S'(\boldsymbol{z})\boldsymbol{v} \in \mathcal{U}$ solves the state sensitivity equation

$$c_{\boldsymbol{u}}(S(\boldsymbol{z}), \boldsymbol{z})\boldsymbol{w} = -c_{\boldsymbol{z}}(S(\boldsymbol{z}), \boldsymbol{z})\boldsymbol{v}, \tag{7}$$

and $\boldsymbol{\pi} = \boldsymbol{\lambda}'(\boldsymbol{z})\boldsymbol{v} \in \mathcal{U}$ solves the adjoint sensitivity equation

$$c_{\boldsymbol{u}}(S(\boldsymbol{z}), \boldsymbol{z})^\top \boldsymbol{\pi} = -L_{\boldsymbol{uu}}(S(\boldsymbol{z}), \boldsymbol{z}, \Lambda(\boldsymbol{z}))\boldsymbol{w} - L_{\boldsymbol{uz}}(S(\boldsymbol{z}), \boldsymbol{z}, \Lambda(\boldsymbol{z}))\boldsymbol{v}. \tag{8}$$

Note that (7) is solved forward in time, while (8) is solved backward in time like (6). Moreover, to apply the Hessian, we generally must store $S(\boldsymbol{z})$, $\Lambda(\boldsymbol{z})$, and $\boldsymbol{w}$. Consequently, we will sketch these quantities for use in Hessian applications.

3 Randomized sketching

In order to leverage randomized sketching to reduce the dimensionality of $\boldsymbol{u} = S(\boldsymbol{z})$ (and similarly for $\boldsymbol{\lambda}$ and $\boldsymbol{w}$), we reshape $\boldsymbol{u}$ as the matrix

$$\mathbf{U} = \left[\, u_1 \mid \cdots \mid u_N \,\right] \in \mathbb{R}^{M \times N}.$$

Given a target rank $r \in \{1, \ldots, \min\{M, N\}\}$, randomized sketching produces a rank-r approximation $\{\!\{\mathbf{U}\}\!\}_r$ of $\mathbf{U}$ that requires only $O(r(M+N))$ storage. Let $k \in \{r, \ldots, N\}$, $s \in \{k, \ldots, M\}$ and define the two random linear dimension reduction maps (DRMs)

$$\boldsymbol{\Psi} \in \mathbb{R}^{s \times M} \qquad \text{and} \qquad \boldsymbol{\Omega} \in \mathbb{R}^{N \times k},$$

which have independent standard normal entries. The sketch of $\mathbf{U}$ consists of two matrices:

$$\begin{aligned} \mathbf{Y} &:= \mathbf{U}\boldsymbol{\Omega} \in \mathbb{R}^{M \times k}, && \text{the range sketch;} \\ \mathbf{W} &:= \boldsymbol{\Psi}\mathbf{U} \in \mathbb{R}^{s \times N}, && \text{the co-range sketch.} \end{aligned}$$

The linearity of the sketching operations allows us to compute $\mathbf{Y}$ and $\mathbf{W}$ as the columns of $\mathbf{U}$ are generated, reducing the storage requirements. See [18, Alg. 2] for details. Note that this specific sketch can be replaced with other sketch algorithms like the one used in [6] and the entries of $\boldsymbol{\Omega}$ and $\boldsymbol{\Psi}$ can be drawn from other distributions than standard normal [18]. Moreover, we can further reduce memory by not storing the random DRMs Ω and Ψ, but rather storing the seed used to generate them.

To reconstruct a rank-r approximation of $\mathbf{U}$ from $\mathbf{Y}$ and $\mathbf{W}$, one first orthogonalizes $\mathbf{Y}$ using the skinny QR factorization, i.e.,

$$\mathbf{Y} := \mathbf{QR}, \qquad \text{where} \qquad \mathbf{Q} \in \mathbb{R}^{M \times k}.$$

Here, we store $\mathbf{Q}$ in place of $\mathbf{Y}$ and discard the triangular matrix $\mathbf{R}$. We then solve the small, well-conditioned least-squares problem

$$\mathbf{X} := (\mathbf{\Psi Q})^{\dagger}\, \mathbf{W} \in \mathbb{R}^{k \times N},$$

which is stored in place of $\mathbf{W}$ [18]. Finally, we construct the rank-r approximation of $\mathbf{U}$ as

$$\{\!\{\mathbf{U}\}\!\}_r := \mathbf{QX} \in \mathbb{R}^{M \times N}.$$

Note that we never form the product $\mathbf{QX}$ as this would require $O(MN)$ storage. Instead, we reconstruct the columns of $\{\!\{\mathbf{U}\}\!\}_r$ as needed for derivative computations using the product

$$u_n \approx \mathbf{Q}x_n \qquad \text{for} \qquad n = 1, \ldots, N,$$

where x_n denotes the nth column of $\mathbf{X}$.

4 Sketched quasi-Newton trust-region method

We now review the trust-region method described in [6], which is a specialization of the method described in [19]. Given the current iterate $\boldsymbol{z}_\ell$, we compute a trial step $\boldsymbol{s}_\ell$ by approximately solving the trust-region subproblem

$$\min_{\boldsymbol{s} \in \mathcal{Z}} \{m_\ell(\boldsymbol{s}) := \tfrac{1}{2}\boldsymbol{s}^\top \boldsymbol{B}_\ell \boldsymbol{s} + \boldsymbol{g}_\ell^\top \boldsymbol{s}\} \qquad \text{subject to} \qquad \|\boldsymbol{s}\| \leq \Delta_\ell,$$

where $\Delta_\ell > 0$ is the trust-region radius, $\boldsymbol{g}_\ell$ is an approximation of the gradient $\nabla F(\boldsymbol{z}_\ell)$ and $\boldsymbol{B}_\ell$ is a secant approximation of the Hessian $\nabla^2 F(\boldsymbol{z}_\ell)$. We compute $\boldsymbol{g}_\ell$ using sketching via [6, Alg. 4.4], which adaptively increases the sketch rank r until the state reconstruction satisfies

$$\|c(\{\!\{\mathbf{U}_\ell\}\!\}_r, \boldsymbol{z}_\ell)\| \leq \kappa_{\text{grad}} \min\{\|\boldsymbol{g}_\ell\|, \Delta_\ell\}. \tag{9}$$

Here, $\kappa_{\text{grad}} > 0$ is a user-specified parameter that is independent of ℓ and $\mathbf{U}_\ell$ denotes the state matrix associated with the current control $\boldsymbol{z}_\ell$.

The randomness in $\{\!\{\mathbf{U}_\ell\}\!\}_r$, inherited from the DRMs Ω and Ψ, induces randomness in the model gradient $\boldsymbol{g}_\ell$. This randomness changes at each iteration and can create instabilities in standard secant approximations due to random variability between subsequent model gradients. In particular, popular methods like BFGS, DFP, PSB and SR1 [20, 21] utilize the gradient difference

$$\boldsymbol{y}_\ell = \boldsymbol{g}_{\ell+1} - \boldsymbol{g}_\ell. \tag{10}$$

However, $\boldsymbol{g}_{\ell+1}$ and $\boldsymbol{g}_\ell$ are computed from states that use different instances of Ω and Ψ, which can lead to instabilities in the secant approximation especially when κ_{grad} is large. To overcome these instabilities, we replace (10) with

$$\boldsymbol{y}_\ell \approx \nabla^2 F(\boldsymbol{z}_\ell)\boldsymbol{s}_\ell. \tag{11}$$

Since the application of $\nabla^2 F(\boldsymbol{z}_\ell)$ to $\boldsymbol{s}_\ell$ requires storing the state, adjoint and state sensitivity trajectories, we instead sketch these trajectories producing an approximate Hessian application that we use to define $\boldsymbol{y}_\ell$ in (11), cf. [6, Alg. A.5]. Replacing the difference of gradients with a Hessian application was used in [17] for applying stochastic secant methods to solve training problems in machine learning. In our numerical experiments, we employ the limited-memory SR1 and BFGS update formulae [20, 22–25], given by

$$\boldsymbol{B}_{\ell+1} = \boldsymbol{B}_\ell + \frac{(\boldsymbol{y}_\ell - \boldsymbol{B}_\ell \boldsymbol{s}_\ell)\,(\boldsymbol{y}_\ell - \boldsymbol{B}_\ell \boldsymbol{s}_\ell)^\top}{(\boldsymbol{y}_\ell - \boldsymbol{B}_\ell \boldsymbol{s}_\ell)^\top \boldsymbol{s}_\ell} \tag{12a}$$

$$\boldsymbol{B}_{\ell+1} = \boldsymbol{B}_\ell - \frac{(\boldsymbol{B}_\ell \boldsymbol{s}_\ell)(\boldsymbol{B}_\ell \boldsymbol{s}_\ell)^\top}{\boldsymbol{s}_\ell^\top (\boldsymbol{B}_\ell \boldsymbol{s}_\ell)} + \frac{\boldsymbol{y}_\ell \boldsymbol{y}_\ell^\top}{\boldsymbol{s}_\ell^\top \boldsymbol{y}_\ell}, \tag{12b}$$

respectively. As is typically done, we update the SR1 secant approximation only if

$$|\boldsymbol{s}_\ell^\top\,(\boldsymbol{y}_\ell - \boldsymbol{B}_\ell \boldsymbol{s}_\ell)| \geq c\|\boldsymbol{s}_\ell\|\|\boldsymbol{y}_\ell - \boldsymbol{B}_\ell \boldsymbol{s}_\ell\|, \tag{13}$$

where $c \in (0,1)$ is a user-specified constant, cf. [25] for additional information. Similarly, we update the BFGS secant approximation only if the curvature condition

$$\boldsymbol{y}_\ell^\top \boldsymbol{s}_\ell \geq c\|\boldsymbol{y}_\ell\|\|\boldsymbol{s}_\ell\| \tag{14}$$

is satisfied. Again, $c \in (0,1)$ is a user specified constant. For more details, see [20, 23].

Algorithm 1 lists the sketched quasi-Newton trust-region algorithm. To shorten the presentation, we employ the algorithms listed in [6] with the understanding that [6, Alg. A.1] performs the simplified sketching procedure described in Section 3. The functions `Sketch`, `SolveState!`, `AdaptiveRankGradient`, `SolveTRSubProblem`, `SolveStateSensitivity`, and `ApplyFixedRankHessian` can be found in [6]. On the other hand, `UpdateSecant` represents an arbitrary secant method like BFGS or SR1. Under standard assumptions, [6, Th. 4.7] demonstrates that Algorithm 1 converges in the sense that

$$\liminf_{\ell\to\infty} \|\boldsymbol{g}_\ell\| = \liminf_{\ell\to\infty} \|\nabla F(\boldsymbol{z}_\ell)\| = 0. \tag{15}$$

5 Numerical results

We demonstrate the performance of Algorithm 1 on a discretized version of the optimal control problem

$$\min_z \frac{1}{2}\int_0^T \int_0^1 \left\{(u(x,t) - u_d(x,t))^2 + \alpha z(x,t)^2\right\} \mathrm{d}x\mathrm{d}t, \tag{16}$$

Algorithm 1 Sketched quasi-Newton trust-region method

Input: Initial control $\boldsymbol{z}_0$, trust-region radius $\Delta_0 > 0$, initial sketch rank $r_0 > 0$, and trust-region hyperparameter set $P = \{\eta_1, \eta_2, \gamma_1, \gamma_2, \Delta_{\max}\}$

$r \leftarrow r_0$ ▷ Set sketch rank
$\{\!\{\mathbf{U}\}\!\}_r \leftarrow \textsc{Sketch}(M, N, \text{rank} = r)$ ▷ Instantiate state sketch
$F_0 \leftarrow \textsc{SolveState!}(\{\!\{\mathbf{U}\}\!\}_r, \boldsymbol{z}_0)$ ▷ Sketch state and evaluate objective
$(\boldsymbol{g}_0, r) \leftarrow \textsc{AdaptiveRankGradient}(\boldsymbol{z}_0, r, \{\!\{\mathbf{U}\}\!\}_r)$ ▷ Approximate gradient
$\ell \leftarrow 0$
while "Not Converged" **do**
  $(\boldsymbol{s}_\ell, \text{pred}_\ell) \leftarrow \textsc{SolveTRSubProblem}(\boldsymbol{g}_\ell, \Delta_\ell)$ ▷ Compute trial step
  $\{\!\{\mathbf{U}\}\!\}_r.\textsc{Initialize!}(M, N, \text{rank} = r)$ ▷ Re-initialize state sketch
  $F_{\ell+1} \leftarrow \textsc{SolveState!}(\{\!\{\mathbf{U}\}\!\}_r, \boldsymbol{z}_\ell + \boldsymbol{s}_\ell)$ ▷ Compute new objective value
  $\rho_\ell = (F_\ell - F_{\ell+1})/ - m_\ell(s_\ell)$ ▷ Compute ratio of reduction
  if $\rho_\ell \geq \eta_1$ **then** ▷ Validate step using ratio of reduction
    $\boldsymbol{z}_{\ell+1} = \boldsymbol{z}_\ell + \boldsymbol{s}_\ell$
  else
    $\boldsymbol{z}_{\ell+1} = \boldsymbol{z}_\ell$
  end if
  if $\rho_\ell \geq \eta_2$ **then** ▷ Update trust-region radius
    $\Delta_{\ell+1} = \min\{\gamma_2 \Delta_\ell, \Delta_{\max}\}$
  else if $\rho_\ell \leq \eta_1$ **then**
    $\Delta_{\ell+1} = \gamma_1 \|\boldsymbol{s}_\ell\|$
  else
    $\Delta_{\ell+1} = \Delta_\ell$
  end if
  $(\boldsymbol{g}_{\ell+1}, r) \leftarrow \textsc{AdaptiveRankGradient}(\boldsymbol{z}_{\ell+1}, r, \{\!\{\mathbf{U}\}\!\}_r)$ ▷ Update gradient
  if $\rho_\ell \geq \eta_1$ **then**
    $\boldsymbol{y}_\ell \leftarrow \textsc{ApplyFixedRankHessian}(\boldsymbol{z}_\ell, \{\!\{\mathbf{U}\}\!\}_r, \boldsymbol{s}_\ell, r, r)$ ▷ Apply sketched Hessian
    $\boldsymbol{B}_{\ell+1} \leftarrow \textsc{UpdateSecant}(\boldsymbol{s}_\ell, \boldsymbol{y}_\ell)$ ▷ Update secant storage
  end if
end while

where $z : (0,1) \times (0,T) \to \mathbb{R}$ is a distributed control, $u : (0,1) \times (0,T) \to \mathbb{R}$ is the weak solution to viscous Burgers' equation [26, 27]

$$\begin{aligned} u_t(x,t) - \nu u_{xx}(x,t) + u_x(x,t)u(x,t) = z(x,t) \quad & x \in (0,1), \quad t \in (0,T), \\ u(0,t) = u(1,t) = 0 \quad & t \in (0,T), \\ u(x,0) = u_0(x) \quad & x \in (0,1), \end{aligned} \tag{17}$$

$u_d : (0,1) \times (0,T) \to \mathbb{R}$ is the target state, $\alpha > 0$ is the control cost, $u_0 : (0,1) \to \mathbb{R}$ is the initial condition, and $\nu > 0$ is the viscosity. In the following experiments, we set $\alpha = 10^{-2}$ and $\nu = 5 \times 10^{-3}$, $T = 1$, $u_0 = 0$, and the data is given by

$$u_d(x,t) = \begin{cases} 0 & \text{if } (x,t) \in (0, 0.5] \times (0,T), \\ 1 & \text{if } (x,t) \in (0.5, 1) \times (0,T). \end{cases} \tag{18}$$

We discretize (17) in space using continuous piecewise linear finite elements on a uniform mesh with $M = 512$ interior vertices and in time using backward Euler with $N = 512$ uniform time steps. For an explicit description of the finite-dimensional problem, see [28]. We apply Algorithm 1 to the discretized reduced optimization problem and update the sketch rank using the rank update function

$$\mu(r, \tau) = \max\left\{r + 1, \left\lceil \frac{-p_2 + \log \tau}{p_1} \right\rceil\right\},$$

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 $p_1 = -0.65$ and $p_2 = 11.56$. To quantify the memory savings achieved by Algorithm 1, we employ the compression factor

$$\zeta := \frac{\text{full storage}}{\text{reduced storage}} = \frac{NM}{(kM + sN)}. \tag{19}$$

As suggested in [18, 29], we choose $k = 2r + 1$ and $s = 2k + 1$. Figure 1 depicts the tail energy and sketching error averaged over 60 realizations of the random DRMs for the uncontrolled and optimal states. Both the tail energy and sketching error decay exponentially fast until saturating below $\mathcal{O}(10^{-12})$ indicating that the sketched state produces an accurate approximation as the rank increases.

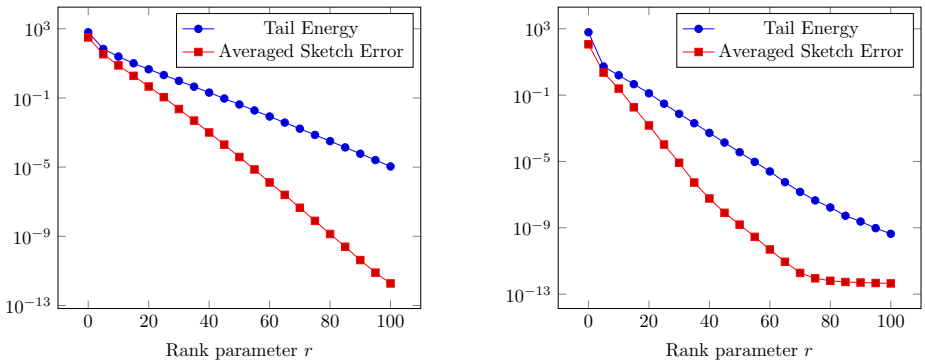

Fig. 1: The sketching error averaged over 60 realizations (red) and the tail energy (blue) for the uncontrolled (left) and the optimal (right) states of the viscous Burgers' equation. For this experiment, $M = N = 512$.

We terminate Algorithm 1 when the relative stopping condition

$$\|\boldsymbol{g}_\ell\| \leq 10^{-4}\|\boldsymbol{g}_0\| \tag{20}$$

is satisfied. Recall that (15) ensures that (20) is satisfied after finitely many iterations. We further set $\Delta_0 = 50$, $\eta_1 = 0.05$, $\eta_2 = 0.75$, $\gamma_1 = 0.25$, $\gamma_2 = 2.5$, $\Delta_{\max} = 10^3$

and $r_0 = 1$. Table 1 demonstrates typical runs of Algorithm 1 using BFGS. Here, we compare the performance of the algorithm when using the traditional gradient difference (10) (denoted **GradDiff**) and the inexact Hessian application (11) (denoted **Hessian**). For these results, we set $\kappa_{\text{grad}} = 10^3$ and notice that both **GradDiff** and **Hessian** perform similarly. However, as we will see, the performance of **GradDiff** degrads as κ_{grad} increases (i.e., when using poor gradient approximations). Tables 2

L	**Method**	**it**	**feval**	**itCG**	**acc**	**max r**	ζ
5	**GradDiff**	60	115	117	41	16	5.12
	Hessian	62	119	126	43	15	5.44
10	**GradDiff**	59	116	125	42	15	5.44
	Hessian	55	107	124	37	16	5.12
20	**GradDiff**	48	93	121	31	15	5.44
	Hessian	50	97	124	32	15	5.44
full	**GradDiff**	52	102	130	35	17	4.83
	Hessian	46	90	117	30	15	5.44

Table 1: Typical performance of Algorithm 1 for solving the optimal control of the viscous Burgers' equation: **L** is the ℓ-BFGS memory size with *full* meaning *full memory*, **it** is the number of trust-region iterations, **feval** is the number of function evaluations, **itCG** is the number of CG iterations, **acc** is the number of accepted steps, **max r** is the final sketch rank, and ζ is the compression factor. For this experiment, $M = N = 512$ and $k_{grad} = 10^3$.

and 3 display the algorithm statistics for **GradDiff** and **Hessian** with increasing κ_{grad}, using BFGS and SR1, respectively. We repeated each run 60 times using different realizations of the random DRMs and report the average results. As demonstrated, **Hessian** tends to outperform **GradDiff**, especially for large κ_{grad} and large secant memory **L**. This improvement is amplified when using SR1. For example, when using full-memory secant approximations (denoted by *full*), we see a significant performance improvement for **Hessian** over **GradDiff**. As emphasized in [17], this result is not surprising since the accumulation of errors in the gradient differences is accentuated by larger memory secant approximations.

6 Extensions

As described, Algorithm 1 applies only to smooth unconstrained problems. However, it is straightforward to modify the algorithm for solving convex-constrained or non-smooth regularized problems. In particular, Algorithm 1 is an instance of the more general algorithm introduced in [30] for solving problems of the form

$$\min_{\boldsymbol{z} \in \mathcal{Z}} \; F(\boldsymbol{z}) + \phi(\boldsymbol{z}),$$

L	Method	it	feval	itCG	acc	max r	ζ
			$\kappa_{\text{grad}} = \mathbf{1000}$				
10	**GradDiff**	57.00	111.87	120.80	39.78	15.73	5.20
	Hessian	54.43	106.47	124.97	36.87	15.87	5.16
20	**GradDiff**	50.45	99.53	122.92	34.10	15.82	5.17
	Hessian	47.92	94.10	127.37	31.18	15.72	5.20
full	**GradDiff**	49.17	97.53	127.20	33.37	15.75	5.19
	Hessian	46.77	92.35	130.92	30.45	16.10	5.08
			$\kappa_{\text{grad}} = \mathbf{10{,}000}$				
10	**GradDiff**	60.43	117.43	123.38	42.63	12.77	6.35
	Hessian	57.68	111.28	131.08	39.48	12.73	6.36
20	**GradDiff**	53.33	104.07	125.22	36.70	12.85	6.31
	Hessian	50.00	96.60	131.22	32.83	12.52	6.47
full	**GradDiff**	50.98	99.70	125.05	34.85	12.67	6.39
	Hessian	49.23	95.10	131.45	32.20	12.58	6.44
			$\kappa_{\text{grad}} = \mathbf{100{,}000}$				
10	**GradDiff**	88.92	168.03	158.90	66.55	10.27	7.80
	Hessian	88.98	168.75	164.40	67.22	10.50	7.64
20	**GradDiff**	84.67	160.23	164.85	63.00	10.42	7.69
	Hessian	76.45	144.78	156.48	56.13	10.42	7.69
full	**GradDiff**	81.60	151.30	166.83	58.93	10.43	7.69
	Hessian	71.33	135.87	154.15	52.28	10.48	7.48

Table 2: Performance statistics averaged over 60 runs using limited-memory BFGS and various κ_{grad}: **L** is the ℓ-BFGS memory size with *full* meaning *full memory*, **it** is average the number of trust-region iteration, **feval** is the average number of function evaluations, **itCG** is the average number of CG iterations, **acc** is the average number of accepted steps, **max r** is the average final rank, ζ is the average compression factor.

where $\phi : \mathcal{Z} \to (-\infty, +\infty]$ is proper, closed and convex. In this more general setting, one changes the trust-region subproblem to account for the potentially nonsmooth term ϕ. This modification necessitates different subproblem solvers `SolveTRSubProblem`, cf. [31]. Moreover, the gradient error criterion (9) changes to

$$\|c(\{\!\{\mathbf{U}_\ell\}\!\}_r, \boldsymbol{z}_\ell)\| \leq \kappa_{\text{grad}} \min\{\tfrac{1}{t}\|\boldsymbol{z}_\ell - \text{prox}_{t\phi}(\boldsymbol{z}_\ell - t\boldsymbol{g}_\ell)\|, \Delta_\ell\},$$

where $t > 0$ is fixed and $\text{prox}_{t\phi}$ denotes the usual proximity operator of ϕ [32]. See [33] for an initial study of randomized sketching in nonsmooth dynamic optimization using sketched Hessian applications for $\boldsymbol{B}_\ell$. Finally, in [6], the authors require that the sequence of model Hessians $\{\boldsymbol{B}_\ell\}$ is uniformly bounded to achieve (15), cf. [6, As. 2.1]. However, we can replace this requirement with the weaker condition

$$\sum_{\ell=0}^{\infty} \frac{1}{1 + \max\limits_{j=0,\ldots,\ell} \|\boldsymbol{B}_j\|} = +\infty.$$

See [30, Th. 3] for additional details.

L	Method	it	feval	itCG	acc	max r	ζ
			κ_{grad} = **1,000**				
10	**GradDiff**	38.53	79.07	163.60	25.63	16.08	5.09
	Hessian	40.47	81.62	167.15	26.18	15.95	5.13
20	**GradDiff**	39.90	80.38	204.80	25.77	16.03	5.11
	Hessian	42.13	85.02	267.00	27.97	16.10	5.08
full	**GradDiff**	41.82	83.68	237.20	26.93	16.18	5.06
	Hessian	45.72	90.40	326.25	30.17	16.63	4.93
			κ_{grad} = **10,000**				
10	**GradDiff**	43.05	85.85	172.85	29.27	12.57	6.44
	Hessian	44.82	87.82	173.82	29.28	12.68	6.39
20	**GradDiff**	50.90	97.92	203.15	33.20	12.68	6.39
	Hessian	47.62	92.75	256.07	31.55	12.95	6.26
full	**GradDiff**	61.78	116.38	298.43	40.72	13.22	6.14
	Hessian	50.95	98.50	340.98	34.00	13.25	6.13
			κ_{grad} = **100,000**				
10	**GradDiff**	86.57	161.95	217.35	62.88	10.50	7.64
	Hessian	69.67	131.83	174.05	50.17	10.50	7.64
20	**GradDiff**	95.15	177.15	242.57	69.15	10.58	7.58
	Hessian	69.27	131.15	192.42	49.47	10.47	7.66
full	**GradDiff**	168.35	305.57	846.60	123.55	14.13	5.76
	Hessian	74.63	140.50	213.45	53.75	10.40	7.71

Table 3: Performance statistics averaged over 60 runs using limited-memory SR1 and various κ_{grad}: **L** is the ℓ-SR1 memory size with *full* meaning *full memory*, **it** is average the number of trust-region iteration, **feval** is the average number of function evaluations, **itCG** is the average number of CG iterations, **acc** is the average number of accepted steps, **max r** is the average final rank, ζ is the average compression factor.

Acknowledgments

This research was sponsored by the U.S. Department of Energy Office of Science and the U.S. Air Force Office of Scientific Research. This article has been authored by an employee of National Technology & Engineering Solutions of Sandia, LLC under Contract No. DE-NA0003525 with the U.S. Department of Energy (DOE). The employee owns all right, title and interest in and to the article and is solely responsible for its contents. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this article or allow others to do so, for United States Government purposes. The DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan.

Declarations. Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

References

[1] Krebs, J.R., Anderson, J.E., Hinkley, D., Neelamani, R., Lee, S., Baumstein, A., Lacasse, M.-D.: Fast full-wavefield seismic inversion using encoded sources. Geophysics **74**(6), 177–188 (2009)

[2] Wang, Q., Moin, P., Iaccarino, G.: Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation. SIAM Journal on Scientific Computing **31**(4), 2549–2567 (2009)

[3] Epanomeritakis, I., Akçelik, V., Ghattas, O., Bielak, J.: A Newton-CG method for large-scale three-dimensional elastic full-waveform seismic inversion. Inverse Problems **24**(3), 034015 (2008)

[4] Arridge, S.R.: Optical tomography in medical imaging. Inverse problems **15**(2), 41 (1999)

[5] Klose, A.D., Hielscher, A.H.: Optical tomography using the time-independent equation of radiative transfer—part 2: inverse model. Journal of Quantitative Spectroscopy and Radiative Transfer **72**(5), 715–732 (2002)

[6] Muthukumar, R., Kouri, D.P., Udell, M.: Randomized sketching algorithms for low-memory dynamic optimization. SIAM Journal on Optimization **31**(2), 1242–1275 (2021)

[7] Hinze, M., Pinnau, R., Ulbrich, M., Ulbrich, S.: Optimization with PDE Constraints vol. 23. Springer, (2008)

[8] Fahl, M., Sachs, E.W.: Reduced order modelling approaches to PDE-constrained

optimization based on proper orthogonal decomposition. In: Large-scale PDE-constrained Optimization, pp. 268–280. Springer, (2003)

[9] Zahr, M.J., Carlberg, K.T., Kouri, D.P.: An efficient, globally convergent method for optimization under uncertainty using adaptive model reduction and sparse grids. SIAM/ASA Journal on Uncertainty Quantification **7**(3), 877–912 (2019)

[10] Leugering, G., Benner, P., Engell, S., Griewank, A., Harbrecht, H., Hinze, M., Rannacher, R., Ulbrich, S.: Trends in PDE Constrained Optimization vol. 165. Springer, (2014)

[11] Aupy, G., Herrmann, J., Hovland, P., Robert, Y.: Optimal multistage algorithm for adjoint computation. SIAM Journal on Scientific Computing **38**(3), 232–255 (2016)

[12] Griewank, A., Walther, A.: Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. ACM Transactions on Mathematical Software (TOMS) **26**(1), 19–45 (2000)

[13] Stumm, P., Walther, A.: New algorithms for optimal online checkpointing. SIAM Journal on Scientific Computing **32**(2), 836–854 (2010)

[14] Tropp, J.A., Yurtsever, A., Udell, M., Cevher, V.: Streaming low-rank matrix approximation with an application to scientific simulation. SIAM Journal on Scientific Computing **41**(4), 2430–2463 (2019)

[15] Halko, N., Martinsson, P.-G., Tropp, J.A.: Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM review **53**(2), 217–288 (2011)

[16] Nakatsukasa, Y., Tropp, J.A.: Fast & accurate randomized algorithms for linear systems and eigenvalue problems. arXiv preprint arXiv:2111.00113 (2021)

[17] Byrd, R.H., Hansen, S.L., Nocedal, J., Singer, Y.: A stochastic quasi-Newton method for large-scale optimization. SIAM Journal on Optimization **26**(2), 1008–1031 (2016)

[18] Tropp, J.A., Yurtsever, A., Udell, M., Cevher, V.: Practical sketching algorithms for low-rank matrix approximation. SIAM Journal on Matrix Analysis and Applications **38**(4), 1454–1485 (2017)

[19] Kouri, D.: A trust-region algorithm with adaptive stochastic collocation for PDE optimization under uncertainty. SIAM Journal on Scientific Computing **35**(4), 1847–1879 (2013)

[20] Nocedal, J., Wright, S.J.: Numerical Optimization. Springer, (1999)

[21] Dennis, J.E. Jr, Moré, J.J.: Quasi-Newton methods, motivation and theory. SIAM

review **19**(1), 46–89 (1977)

[22] Byrd, R.H., Nocedal, J., Schnabel, R.B.: Representations of quasi-Newton matrices and their use in limited memory methods. Mathematical Programming **63**(1-3), 129–156 (1994)

[23] Morales, J.L.: A numerical study of limited memory BFGS methods. Applied Mathematics Letters **15**(4), 481–487 (2002)

[24] Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Mathematical programming **45**(1-3), 503–528 (1989)

[25] Byrd, R.H., Khalfan, H.F., Schnabel, R.B.: Analysis of a symmetric rank-one trust region method. SIAM Journal on Optimization **6**(4), 1025–1039 (1996)

[26] Burgers, J.M.: Hydrodynamics—Application of a Model System to Illustrate Some Points of the Statistical Theory of Free Turbulence. Springer, (1995)

[27] Burgers, J.M.: A mathematical model illustrating the theory of turbulence. Advances in applied mechanics **1**, 171–199 (1948)

[28] Heinkenschloss, M.: Numerical solution of implicitly constrained optimization problems. Technical report, Rice University (2008)

[29] Sun, Y., Guo, Y., Luo, C., Tropp, J., Udell, M.: Low-rank Tucker approximation of a tensor from streaming data. SIAM Journal on Mathematics of Data Science **2**(4), 1123–1150 (2020)

[30] Baraldi, R.J., Kouri, D.P.: A proximal trust-region method for nonsmooth optimization with inexact function and gradient evaluations. Mathematical Programming **201**(1-2), 559–598 (2023)

[31] Baraldi, R.J., Kouri, D.P.: Efficient proximal subproblem solvers for a nonsmooth trust-region method. Optimization Online (2023)

[32] Bauschke, H.H., Combettes, P.L.: Convex Analysis and Monotone Operator Theory in Hilbert Spaces. CMS Books in Mathematics. Springer, (2017)

[33] Baraldi, R.J., Herberg, E., Kouri, D.P., Antil, H.: Adaptive randomized sketching for dynamic nonsmooth optimization. In: Society for Experimental Mechanics Annual Conference and Exposition, pp. 107–116 (2023). Springer