Available online at www.sciencedirect.com

ScienceDirect

Journal of Computational Physics 227 (2008) 6184–6205

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

A Monte Carlo method for solving unsteady adjoint equations

Qiqi Wang [a,*], David Gleich [a], Amin Saberi [a], Nasrollah Etemadi [b], Parviz Moin [b]

[a] *Institute of Computational and Mathematical Engineering, Stanford University, Stanford, CA 94305, USA*
[b] *Center for Turbulence Research, Stanford University, Stanford, CA 94305, USA*

Received 2 June 2006; received in revised form 30 October 2007; accepted 2 March 2008
Available online 17 March 2008

Abstract

Traditionally, solving the adjoint equation for unsteady problems involves solving a large, structured linear system. This paper presents a variation on this technique and uses a Monte Carlo linear solver. The Monte Carlo solver yields a forward-time algorithm for solving unsteady adjoint equations. When applied to computing the adjoint associated with Burgers' equation, the Monte Carlo approach is faster for a large class of problems while preserving sufficient accuracy.
© 2008 Elsevier Inc. All rights reserved.

Keywords: Adjoint equation; Unsteady adjoint equation; Monte Carlo linear solver

1. Introduction

Adjoint based methods are widely used tools for analyzing and controlling the behavior of many systems. In these methods, one models a system as an algebraic or differential equation, then the adjoint equation of the original problem is derived, and finally both equations are solve. From the solution of the adjoint equation, the derivative of an objective function is calculated with respect to a set of control parameters. Often this procedure is called backward algorithmic (automatic) differentiation [15]. Adjoint based methods solve a large class of optimization problems, inverse problems and control problems [2,5,11,16,17,20].

Another important application of adjoint based methods is posterior error estimation. In this context, these methods are often called duality based methods [12,13,18]. They are particularly useful when the original problem is a discretized partial differential equation. When the numerical error caused by the discretization is high, we can apply automatic adaptive mesh refinement [19] based on the adjoint solution to reduce the numerical error at a fixed computational cost.

Although adjoint based methods have a long history in computational sciences and engineering [3], computing the solution of an adjoint equation is difficult: when the original problem is a time dependent (unsteady)

[*] Corresponding author. Tel.: +1 650 796 4472.
E-mail address: qiqi@stanford.edu (Q. Wang).

0021-9991/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2008.03.006

problem, solving its adjoint equation is a backward-time procedure and requires the full trajectory to be stored in memory. The trajectory is formed by the solution of the original problem at all time steps, and is often too large to store in memory. In [14], Griewank proposed a very interesting iterative checkpointing scheme called revolve for dealing with this problem. In his scheme, storing the full trajectory is avoided by iteratively solving the original problem. This idea made solving adjoint equations possible for larger unsteady problems. Since then, a number of similar schemes have been proposed [4,28]. Nevertheless, all of these schemes are significantly more expensive than solving the original problem in terms of both memory requirements and computational time. As a result, if the original problem is very large in terms of the number of degrees of freedom, solving the adjoint equation is still prohibitively expensive.

In this paper, we propose a new method for solving the adjoint equation for unsteady problems. Instead of computing an exact solution, we use a Monte Carlo method to approximate the solution. This method samples Markovian random walks in the space–time structure of the original problem and estimates quantities of interest from these samples. The method builds upon Monte Carlo linear solvers for general linear systems [10,7,27,22] and some related works [23]. We found that the Monte Carlo method is particularly suitable for solving unsteady adjoint equations. In contrast to traditional methods used for solving the adjoint equation, this method is a forward-time procedure. Amazingly, neither storing the trajectory nor iteratively solving the original problem is required. Therefore, the memory requirement and computational time of our Monte Carlo method are a constant multiples of the original problem (as opposed to exact solution methods where these requirements are $\log m$ times larger, where m is the number of time steps).

In the remainder of this paper, Section 2 introduces the unsteady adjoint equation. Section 3 describes the traditional exact solution methods. In Section 4, we introduce our Monte Carlo method for solving adjoint equations. Because many large problems arise from discretized partial differential equations, we describe our method specifically for this case. The error and variance of this method is analyzed in Section 5. In Sections 6 and 7, we show the results of several numerical experiments with Burgers' equation.

2. The unsteady adjoint equation

Consider a state vector u controlled by a control vector η via constraint or governing equation $\mathcal{R}$. The objective function $\mathcal{F}$ is defined on the space of u and η. We denote this by

$$\begin{cases} \mathcal{F}(u,\eta) \\ \mathcal{R}(u,\eta) = 0, \end{cases} \tag{1}$$

where $\mathcal{F}$ is a scalar function and $\mathcal{R}$ is a vector function with $\dim(\mathcal{R}) = \dim(u)$. In the context of an adjoint equation, system (1) is often referred as the original problem.

Many engineering applications fit this problem type. Generally, $\mathcal{R}$ is an algebraic or differential equation modeling a physical system, u is a vector describing the state of the system and η is a vector composed of a set of control parameters. In the context of computational fluid dynamics, most problems concern objects in a flow field. In these problems, the constraint $\mathcal{R}$, the Navier–Stokes equation, controls the flow field u. We are often interested in objective functions such as the lift, drag and other forces, which are possible candidates for $\mathcal{F}$. The control parameters might be the geometry of the object itself or perturbations to the boundary conditions.

The adjoint equation of the system (1) is defined as the linear system

$$\left(\frac{\partial \mathcal{R}}{\partial u}\right)^{\mathrm{T}} \psi = \left(\frac{\partial \mathcal{F}}{\partial u}\right)^{\mathrm{T}}, \tag{2}$$

where $(\frac{\partial \mathcal{R}}{\partial u})^{\mathrm{T}}$ is the constraint Jacobian and ψ is the adjoint solution.

The adjoint equation is widely used in analyzing and controlling the system (1). For example, many optimization problems, inverse problems and control problems require computing the derivative of the objective function with respect to the control parameters of the original problem (1),

$$\frac{\mathrm{d}\mathcal{F}(u(\eta),\eta)}{\mathrm{d}\eta} = \frac{\partial \mathcal{F}}{\partial \eta} + \frac{\partial \mathcal{F}}{\partial u}\frac{\mathrm{d}u(\eta)}{\mathrm{d}\eta},$$

where $u(\eta)$ is an implicit function defined by $\mathcal{R}$ and its derivative $\frac{du(\eta)}{d\eta}$ can be obtained from

$$0 = \frac{d\mathcal{R}(u(\eta),\eta)}{d\eta} = \frac{\partial\mathcal{R}}{\partial\eta} + \frac{\partial\mathcal{R}}{\partial u}\frac{du(\eta)}{d\eta}.$$

Therefore, with some manipulation of the adjoint equation (2), we get

$$\frac{d\mathcal{F}(u(\eta),\eta)}{d\eta} = \frac{\partial\mathcal{F}}{\partial\eta} - \frac{\partial\mathcal{F}}{\partial u}\left(\frac{\partial\mathcal{R}}{\partial\eta}\right)^{-1}\frac{\partial\mathcal{R}}{\partial\eta} = \frac{\partial\mathcal{F}}{\partial\eta} - \psi^{\mathrm{T}}\frac{\partial\mathcal{R}}{\partial\eta}, \tag{3}$$

which is a linear function of the adjoint solution. In this paper, we focus on the case when the original problem (1) is an unsteady problem; e.g. when the constraint $\mathcal{R}$ is a time dependent partial differential equation. In this case, we can order the elements of the state vector and the constraint by time steps, i.e.

$$u = (u^{(1)},\ldots,u^{(m)})^{\mathrm{T}}, \quad \mathcal{R} = (\mathcal{R}^{(1)},\ldots,\mathcal{R}^{(m)})^{\mathrm{T}},$$

where m is the number of time steps, $u^{(i)}$ and $\mathcal{R}^{(i)}$ are the state vector and the constraint at time step i. With this ordering, the matrix $(\frac{\partial\mathcal{R}}{\partial u})^{\mathrm{T}}$ in the unsteady adjoint equation (2) is well structured. This is because for unsteady problems, $\mathcal{R}^{(i)}$ only depends on the part of u up to time step i. In other words, a block of the constraint Jacobian $J_{ij} \neq 0$ only if $j \leqslant i$. As a result, the Jacobian matrix $\frac{\partial\mathcal{R}}{\partial u}$ is in block-lower-triangular form.

$$\frac{\partial\mathcal{R}}{\partial u} = \begin{bmatrix} J_{11} & & & \\ J_{21} & J_{22} & & \\ \ddots & \ddots & \ddots & \\ J_{m1} & \ddots & J_{mm-1} & J_{mm} \end{bmatrix}, \tag{4}$$

where each block describes the spatial dependence of the constraint.

$$J_{ts} = \left(\frac{\partial\mathcal{R}^{(t)}}{\partial u^{(s)}}\right).$$

The adjoint equation (2) now becomes

$$\begin{bmatrix} J_{11}^{\mathrm{T}} & J_{21}^{\mathrm{T}} & \ddots & J_{m1}^{\mathrm{T}} \\ & J_{22}^{\mathrm{T}} & \ddots & \ddots \\ & & \ddots & \ddots \\ & & & J_{mm}^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \psi^{(1)} \\ \psi^{(2)} \\ \vdots \\ \psi^{(m)} \end{bmatrix} = \begin{bmatrix} b^{(1)} \\ b^{(2)} \\ \vdots \\ b^{(m)} \end{bmatrix}, \tag{5}$$

where

$$b^{(t)} = \left(\frac{\partial\mathcal{F}}{\partial u^{(t)}}\right)^{\mathrm{T}}.$$

In this representation, the adjoint solution ψ is split into m parts. We call $\psi^{(t)}$ the adjoint solution at time step t.

Moreover, if $\mathcal{R}$ comes from a discretized unsteady differential equations, the Jacobian matrix is also block-banded. In this case, $\mathcal{R}^{(t)}$ only depends on the part of u that is in a neighborhood of time step t, and $J_{ts} \neq 0$ only if s is in a neighborhood of t. Hence the Jacobian matrix has a block-bandwidth that depends on the temporal discretization scheme. For example, if the original problem uses a one-step scheme, $\mathcal{R}^{(t)}$ depends only on $u^{(t)}$ and $u^{(t-1)}$. In this case, the block-bandwidth is two. If it is a two-step scheme, then the block-bandwidth is three, etc. In particular, if the differential equation is discretized using an explicit scheme, then $\mathcal{R}^{(t)} = u^{(t)} - f(u^{(t-1)},\ldots)$, and the diagonal blocks of the Jacobian are identity matrices

$$J_{tt} = \left(\frac{\partial\mathcal{R}^{(t)}}{\partial u^{(t)}}\right) = I.$$

Note that in this case, the entire Jacobian matrix is lower-triangular instead of just block-lower-triangular.

In addition to the fixed block-bandwidth, each block of the Jacobian matrix is sparse when the original problem is a discretized partial differential equation. This fact follows because in most spatial discretization schemes, the constraint $\mathcal{R}$ at a mesh-point only depends on its neighboring mesh-points. Denote

$$u^{(t)} = (u_1^{(t)}, \dots, u_n^{(t)})^{\mathrm{T}}, \quad \mathcal{R}^{(t)} = (\mathcal{R}_1^{(t)}, \dots, \mathcal{R}_n^{(t)})^{\mathrm{T}},$$

where $\mathcal{R}_i^{(t)}$ and $u_i^{(t)}$ are the constraint residue and state variable at mesh-point i and time step t. Then $\frac{\partial \mathcal{R}_i^{(t)}}{\partial u_j^{(s)}}$ is non-zero only if j is a neighbor mesh-point of i. Therefore, each block of the constraint Jacobian

$$J_{ts} = \begin{bmatrix} \frac{\partial \mathcal{R}_1^{(t)}}{\partial u_1^{(s)}} & \cdots & \frac{\partial \mathcal{R}_1^{(t)}}{\partial u_n^{(s)}} \\ \vdots & & \vdots \\ \frac{\partial \mathcal{R}_n^{(t)}}{\partial u_1^{(s)}} & \cdots & \frac{\partial \mathcal{R}_n^{(t)}}{\partial u_n^{(s)}} \end{bmatrix}$$

is an n by n sparse matrix, where n is the number of grid points in space.

Thus far, we have studied the adjoint as the solution of a linear system – a well structured sparse system if the original problem is a discretized PDE in space and time. One thing that distinguishes the unsteady adjoint equation from other linear systems is its huge size. In an unsteady PDE, the size of the adjoint linear system is $N = n \cdot m$. Where n is proportional to the number of spatial mesh points (up to billions) and m is the number of time steps (tens of thousands). For many recently attempted computational fluid dynamic problems, the size of the adjoint equation is tens of trillions. As a result, solving adjoint equations requires different strategies and techniques than solving general linear systems.

One direct consequence of such a huge size is the unavailability of required memory to store the entire linear system. Therefore, it is necessary to use a piece by piece approach to solve the adjoint equation. In other words, the solution of the adjoint equation must be computed one piece at a time, based on the information of the adjoint equation given one piece at a time. For this reason, the matrix ordering is a key consideration in developing an algorithm for solving unsteady adjoint equations. Another consideration is how the adjoint solution will be used. Often applications only need part of the solution, or a linear function of the solution. Algorithms that directly compute these final quantities without commuting the full solution should be computationally economical. Note that in this case, we generalize the term "solving adjoint equation" to computing linear functions of the adjoint solution without computing the full solution.

3. Exact solution of adjoint equations

The structure of the adjoint equation, as shown in Eq. (5), makes the piece by piece strategy possible. One approach is to utilize its block-lower-triangular structure and solve using block-back-substitution. In the block-back-substitution, we first solve for $\psi^{(m)}$, then solve for $\psi^{(m-1)}$, and finally solve for $\psi^{(1)}$. This approach for solving the exact solution of unsteady adjoint equation is consistent with the algorithm of backward automatic differentiation [15].

This way of solving the unsteady adjoint equation is a time-reverse algorithm. When solving (5) using block-back-substitution, the first step is to solve for $\psi^{(m)}$ using block J_{mm}^{T} of the matrix and $b^{(m)}$ of the right hand side. Since the J_{mm}^{T} and $b^{(m)}$ depend on $u^{(m)}$, the first step of solving the adjoint equation requires the solution of the original problem at the last time step. As the block backward substitution continues, the solution of the original problem is required in a time-reverse order. As a result, for this simple method to work efficiently, the solution of the original problem at all time steps, namely the full trajectory, must be stored in memory.

However, for even moderately large problems, the memory required to store the full trajectory is too large. One solution to this problem is to use a checkpointing scheme [14,15,28]. These methods are based on the idea of "divide and conquer". Although the whole trajectory cannot be stored, we can speed up the computation by restarting from a set of checkpoints. Additional forward iterations of the original problem move the solution between checkpoints [4].

In 1992, Griewank [14] first proposed the scheme revolve which uses this idea to achieves optimal logarithmic behavior in terms of both computational time and memory requirement. Revolve and many other checkpointing schemes proposed since then use $O(\log m)$ times the memory and computational time of the original problem. Using these schemes, the cost of solving the adjoint equation is only a relatively small factor times the cost of solving the original problem, even if the number of time steps is large.

4. Monte Carlo method for adjoint equation

In this section, we propose a Monte Carlo method for solving the unsteady adjoint equation. Both the memory requirement and computational time of this scheme are $O(1)$ times that of the original problem, independent of the number of time steps of the original problem. As demonstrated in Section 7, our Monte Carlo method has better scaling efficiency than the optimal exact solution method.

Monte Carlo methods have already been shown to be efficient for solving many systems of linear equations, especially when the system is very large and the required precision is relatively low [27,7,1]. These methods craft statistical estimators whose mathematical expectation is a component of the solution vector. Random sampling of these estimators yields approximate solutions [27,24,29]. The main ideas of these methods were proposed by von Neumann and Ulam and are extended by Forsythe and Liebler [10].

Using Monte Carlo methods has several known advantages for solving linear equations. First, the computational cost of obtaining one component of the solution vector using these methods is independent of the size of the linear system [27]. More precisely, it is $O(ql)$, where q is the number of random walks and l is their length. Both q and l are independent of the size of the linear system, and can be controlled to obtain any desired precision. Also, Monte Carlo methods are known for their parallel nature. It is often very easy to parallelize them in a coarse grained manner. Even early in 1949, Metropolisand and Ulam [21] noticed the parallelism inherent in this method.

In addition to these advantages, Monte Carlo methods are particularly suitable for solving the unsteady adjoint equation for two reasons. One is that the Monte Carlo method used in this paper to solve the unsteady adjoint equation is a forward-time procedure. In this procedure, we only use the information necessary to advance to the next time step, which enables us to solve the adjoint equation at the same time as we solve the original equation. Thus we do not need to store full trajectory, neither do we need to iteratively resolve the original problem. This is one great advantage over the traditional method. Secondly, we can directly compute the inner product of the solution ψ with a given vector c, without computing ψ. And the cost of computing $c^T\psi$ using this method is only $O(ql)$, which is the same as the cost of computing one component of the solution vector. In computing $\frac{d\mathcal{F}}{d\eta}$ using Eq. (3) we can take advantage of this by representing $\psi^T\frac{\partial\mathcal{R}}{\partial\eta}$ as the inner product of ψ with $\dim(\eta)$ given vectors

$$\psi^T\frac{\partial\mathcal{R}}{\partial\eta} = \left(\psi^T\frac{\partial\mathcal{R}}{\partial\eta_1}, \psi^T\frac{\partial\mathcal{R}}{\partial\eta_2}, \dots, \psi^T\frac{\partial\mathcal{R}}{\partial\eta_\xi}\right), \quad \text{where } \xi = \dim(\eta)$$

When the dimension of the control vector $\dim(\eta)$ is much smaller than the dimension of the state vector $\dim(u)$, we can save a lot of computational cost by directly computing $\psi^T\frac{\partial\mathcal{R}}{\partial\eta}$. This is useful in applications such as wall control for drag reduction [5,2].

In the remainder of this section, Section 4.1 introduces the (preconditioned) Neumann series representation of the solution. In Section 4.2, we construct the Markovian random walk and the D estimator. We prove that the D estimator is an unbiased estimator to the Neumann series representation of the solution. These discussions are valid for general linear systems and are presented in more detail in [22]. Section 4.3 describes our Monte Carlo algorithm designed specifically for solving unsteady adjoint equations using the D estimator. Section 5.2 discusses the choice of transition probabilities of the Markovian random walk based on the theory of minimum probable error [6]. Section 5.4 discusses choice of preconditioner used in Neumann series representation for our Monte Carlo method. Finally, in Section 6 we fully specify our Monte Carlo algorithm used for Burgers' equation.

4.1. Neumann series representation

To simplify notation, let

$$\bar{A} = \left(\frac{\partial \mathcal{R}}{\partial u}\right)^{\mathrm{T}} \quad \text{and} \quad \bar{b} = \left(\frac{\partial \mathcal{F}}{\partial u}\right)^{\mathrm{T}}$$

in adjoint equation (2). The adjoint equation simplifies to

$$\bar{A}\psi = \bar{b}.$$

We multiply both sides by a block-diagonal preconditioner matrix P, that is easy to invert and preserves the block-upper-triangular structure of the Jacobian $\bar{A}$. Denote

$$A = I - P\bar{A} \quad \text{and} \quad b = P\bar{b}. \tag{6}$$

We note that A has the same block-upper-triangular and block-banded structure of $\bar{A}$. Now the adjoint equation becomes

$$\psi = A\psi + b. \tag{7}$$

The solution ψ to the equation above can be expanded in a Neumann series:

$$\psi = b + Ab + A^2b + A^3b + \cdots. \tag{8}$$

The Neumann series converges and (8) is valid if and only if the spectral radius of A is less than one. When it converges, the Neumann series (8) is the solution of the adjoint equation (7). Note that the inner product of ψ with a given vector c can be represented as

$$c^{\mathrm{T}}\psi = c^{\mathrm{T}}(b + Ab + A^2b + A^3b + \cdots) = \sum_{k=0}^{\infty} c^{\mathrm{T}}A^k b \tag{9}$$

The Monte Carlo method in this paper is a random sampling of this infinite series by a Markovian random walk.

4.2. Markovian random walk and D estimator

For a given vector c, we use Markovian random walks to create random samples of an estimator D introduced in (18) whose mathematical expectation are the inner product of the solution ψ with the given vector c, i.e. $\mathbf{E}(D) = c^{\mathrm{T}}\psi$. In particular, when $c = e_i$, $\mathbf{E}(D) = \psi^{\mathrm{T}}e_i = \psi_i$, and we compute a component of the solution vector. The Markovian random walks have a finite state space with size $N+1$, where N is the size of the adjoint equation. If we label the states as $1, 2, \ldots, N+1$, each of the first N states correspond to a component of the adjoint equation. The special state, a final exit state, gets label $N+1$. The random walks begin from a birth probability r, and follow a transition probability p, where $p(i,j)$ is the transition probability from state i to state j. r and p must satisfy the following conditions [22]:

$$(1) \quad 0 \leqslant r(j), \quad p(i,j) \leqslant 1 \quad \text{for all } i,j, \tag{10}$$

$$(2) \quad \sum_{j=1}^{N+1} p(i,j) = 1 \quad \text{for all } i, \tag{11}$$

$$(3) \quad \sum_{i=1}^{N} r(i) = 1, \tag{12}$$

$$(4) \quad p(N+1, N+1) = 1, \tag{13}$$

$$(5) \quad p(i,j) \neq 0 \Longleftrightarrow A_{ij} \neq 0 \quad \text{and} \quad r(i) \neq 0 \Longleftrightarrow c_i \neq 0, \tag{14}$$

where A_{ij} is the i,jth entry of the matrix A, and c_i is the ith component of the vector c In the four conditions above, the first three define r and p as birth and transition probabilities. The fourth condition defines the state $N+1$ as the final exit state. We note that the probability that the Markovian random walk transits from state j

to the final exit state is $p(j, N+1) = 1 - \sum_{i=1}^{N} p(i,j)$. Once it reaches the final exit state, it always stays there with probability 1. When this happens, we say that the random walk was absorbed in state j. The last condition means that the random walks always stay tied to the matrix, which allows us to go from the random walk model described by p and a to the Neumann series (9) that involves A_{ij} and c_i.

Now we will relate the random walk to the components of the matrix A. Indeed, let the Markov chain be

$$\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_n, \ldots)$$

and

$$\mathbf{i} = (i_0, i_1, \ldots, i_l, N+1, N+1, \ldots), \quad 1 \leqslant i_j \leqslant N$$

be a typical path of the Markov chain that begins at state i_0 and is absorbed at state i_l. The probability that the Markov chain takes this path is

$$\mathcal{P}(\alpha = \mathbf{i}) = r(i_0)p(i_0 i_1)p(i_1 i_2)\cdots p(i_{l-1} I_l)p(i_l, N+1).$$

We define

$$w_{ij} = \begin{cases} \frac{A_{ij}}{p(i,j)} & \text{if } p(i,j) \neq 0 \\ 0 & \text{if } p(i,j) = 0 \end{cases}$$

and as in [22], we define the weight W_k as a random variables on the space of random walks α:

$$W_k(\alpha) = \frac{c_{\alpha_0}}{r(\alpha_0)} \prod_{j=1}^{k} w_{\alpha_j \alpha_{j-1}}, \quad 0 \leqslant k \leqslant l \tag{15}$$

The following proposition gives us insight on why we define the weight in this way.

Proposition 4.1. *Assume conditions (10)–(14) are satisfied. Denote $(\cdot)_j$ as the jth component of a vector, and $(\cdot)_{ij}$ as the i, j entry of a matrix. Then*

$$\mathbf{E}(W_k I_{\{\alpha_k = j\}}) = (c^{\mathrm{T}} A^k)_j \tag{16}$$

In particular, if $c = e_i$,

$$\mathbf{E}(W_k I_{\{\alpha_k = j\}}) = (A^k)_{ij} \tag{17}$$

Proof 4.2. Since $W_0 = \frac{c_{\alpha_0}}{r(\alpha_0)}$ and $\mathcal{P}(\alpha_0 = j) = r(j)$,

$$\mathbf{E}[W_0 I_{\{\alpha_0 = j\}}] = \mathcal{P}(\alpha_0 = j)\frac{c_j}{r(j)} = c_j$$

So (16) holds for $k = 0$. Assume it holds for certain k, we prove it holds for $k+1$.

$$\mathbf{E}[W_{k+1} I_{\{\alpha_{k+1} = j\}}] = \mathbf{E}\left[\sum_i (I_{\{\alpha_k = i, \alpha_{k+1} = j\}} W_k w(i,j))\right] = \sum_i \mathbf{E}[I_{\{\alpha_k = i, \alpha_{k+1} = j\}} W_k]\frac{A_{ij}}{p(i,j)}$$

By using the tower property of conditional expectations, “taking out what is known” and applying Markov property, we have

$$\mathbf{E}[I_{\{\alpha_k = i, \alpha_{k+1} = j\}} W_k] = \mathbf{E}[I_{\{\alpha_k = i\}} W_k]p(i,j).$$

Therefore, by the induction hypothesis,

$$\mathbf{E}[W_{k+1} I_{\{\alpha_{k+1} = j\}}] = \sum_i \mathbf{E}[I_{\{\alpha_k = i\}} W_k]A_{ij} = \sum_i (c^{\mathrm{T}} A^k)_i A_{ij} = (c^{\mathrm{T}} A^{k+1})_j.$$

Thus we conclude that (16) holds true for all $k \geqslant 0$. (17) follows trivially. □

This tells us that $W_k I_{\{\alpha_k = j\}}$ is in fact a randomly sparsified version of vector $c^{\mathrm{T}} A^k$. The randomly sparsified vector contains only one non-zero entry. In every step of the random walk, $W_k I_{\{\alpha_k = j\}}$ is multiplied by A, and further sparsified. We can think of it as the sparsified version of $c^{\mathrm{T}} A^k$ is multiplied by A, by which we get an approximation of $c^{\mathrm{T}}A^{k+1}$, and then sparsify it. Because the Neumann series (9) gives us a relationship between $c^{\mathrm{T}}\psi$ and $c^{\mathrm{T}}A^k$, we can use this relationship to build an estimator for $c^{\mathrm{T}}\psi$ in terms of $W_k I_{\{\alpha_k=j\}}$, which is a randomly sparsified version of $c^{\mathrm{T}}A^k$.

Definition 4.3. Define the D estimator by

$$D(\alpha) = \sum_{k=0}^{\infty} W_k(\alpha) b_{\alpha_k} \tag{18}$$

where $\alpha = (\alpha_0, \alpha_1, \ldots)$ is the random walk; $(b_1, \ldots, b_N)^{\mathrm{T}}$ is the right hand side of equation (7) and $b_{N+1} = 0$.

Now we prove the main theorem that supports the Monte Carlo method.

Theorem 4.4. *Assume that the Neumann series (8) converges for $|A|$, and conditions (10)–(14) are satisfied. Then the expectation of the D estimator*

$$\mathbf{E}(D(\alpha)) = c^{\mathrm{T}}\psi$$

where ψ is the solution of Eq. (7).

Proof 4.5. Since $b_{\alpha_k} = \sum_j I_{\{\alpha_k=j\}}$, the expectation of the D estimator can be represented as

$$\mathbf{E}[D] = \mathbf{E}\left[\sum_{k=0}^{\infty} W_k b_{\alpha_k}\right] = \mathbf{E}\left[\sum_{k=0}^{\infty}\sum_j W_k I_{\{\alpha_k=j\}} b_j\right] = \sum_{k=0}^{\infty}\sum_j \mathbf{E}\left[W_k I_{\{\alpha_k=j\}}\right] b_j.$$

The condition that the Neumann series converges for $|A|$ justifies the exchange of infinite sum and expectation by dominated convergence theorem. It follows from Proposition 4.1 that

$$\mathbf{E}[D] = \sum_{k=0}^{\infty}\sum_j (c^{\mathrm{T}}A^k)_j b_j = \sum_{k=0}^{\infty} c^{\mathrm{T}}A^k b,$$

which is the Neumann series expansion (9). Thus we have

$$\mathbf{E}[D] = c^{\mathrm{T}}\psi, \tag{19}$$

i.e. D is an unbiased estimator of $c^{\mathrm{T}}\psi$. $\square$

This theorem suggests that we can use Monte Carlo method based on the D estimator to approximate $c^{\mathrm{T}}\psi$

$$c^{\mathrm{T}}\psi \approx \frac{1}{q}\sum_{p=1}^{q} D(\alpha[p]),$$

where $\alpha[p]$, $1 \leqslant p \leqslant q$ are independent identically distributed random walks. And the following corollary is a direct consequence of Theorem 4.4 and the strong law of large number.

Corollary 4.6. *Under the conditions of Theorem 4.4, the estimated solution by the Monte Carlo method*

$$\frac{1}{q}\sum_{p=1}^{q} D(\alpha[p]) \to c^{\mathrm{T}}\phi$$

almost surely as $q \to \infty$.

This result justifies our Monte Carlo approach for solving adjoint equation by saying that as the number of random walks increases, the solution of the Monte Carlo method asymptotically converges to the exact solution.

4.3. Monte Carlo algorithm

In this section, we explain the Monte Carlo algorithm for solving the unsteady adjoint equation based on the D estimator constructed in the last section.

We note that condition (14) of the transition probability matrix guarantees that $\mathbf{P}$ has the same block-upper-triangular and block-banded structure as matrix A. We remember that the components of u are ordered by time steps. As a result, random walks defined by this transition probability matrix can only possibly walk to later time steps (upper-diagonal blocks of $\mathbf{P}$ are non-zeros), or walk within the same time step (diagonal blocks of $\mathbf{P}$ are non-zero), but never walk backwards to previous time steps (lower-diagonal blocks of $\mathbf{P}$ are always zero). Therefore, the Markovian random walks only go forward in time. This makes the following forward-time Monte Carlo algorithm possible.

In this algorithm, we generate q independent identically distributed random walks $\alpha[p],\ 1 \leqslant p \leqslant q$. Each of them has transition probabilities $p(i,j)$ and birth probabilities $r(i)$ that satisfies the conditions (10)–(14). We will discuss the choices for r and p in the next section. We denote $\alpha_k[p]$ as the current position of random walk $\alpha[p]$, and we say that a random walk $\alpha[p]$ is at time step t if $\alpha_k[p]$ is in the range of indices that represent time step t. Let $W[p]$ denote $W_k(\alpha[p])$, the weight of random walk at current step. $D[p]$ stores the accumulative sum of the D estimator (18), which equals to $D(\alpha)$ after the random walk is absorbed.

(1) For each $1 \leqslant p \leqslant q$, choose $\alpha_0[p]$ randomly by birth probability vector r, and initialize

$$W[p] = c_{\alpha_0[p]}/r(\alpha_0[p]); \quad D[p] = W[p]b_{\alpha_0[p]}.$$

Then start from $t = 1$, do step 2–4, until done with the last time step $t = m$.

(2) Solve the original problem at time step t, which enables us to compute the corresponding blocks of matrix A and p.

(3) For each random walk $\alpha_{[}p]$ that is at time step t, choose its next state $\alpha_{k+1}[p]$ randomly by transition probabilities p. If $\alpha_{k+1}[p]$ is not the final exit state, update

$$W[p] = W[p]w_{\alpha_k[p]\alpha_{k+1}[p]}; \quad D[p] = D[p] + W[p]b_{\alpha_{k+1}[p]}.$$

If $\alpha_{k+1}[p]'$ is the final exit state, the random walk is absorbed and we freeze $D[p]$.

(4) Repeat step 3 until all random walks at time step t are either absorbed or left the time step. If $t < m$, then let $t = t + 1$ and go to step 2.

(5) After done with the last time step m, all random walks are absorbed. Compute the sample mean of the estimators $\frac{1}{q}\sum_{p=1}^{q} D[p]$, which is our approximation to $c^{\mathrm{T}}\psi$.

It is clear that this is a forward-time algorithm in which we only need to store the current time steps of the original problem and no iterations are needed. Further, it directly yields the inner product of the solution with the given vector. Indeed, there is no difficulty with this algorithm to solve multiple linear functions of the solution vector at the same time. This property is useful in many adjoint based methods. For example, in computing $\frac{d\mathcal{F}(u(\eta),\eta)}{d\eta}$ using formula (3), we can use the Monte Carlo method to directly compute $\psi^{\mathrm{T}}\frac{\partial \mathcal{R}}{\partial \eta}$, which is the inner product of the adjoint solution ψ with $\frac{\partial \mathcal{R}}{\partial \eta_i},\ i = 1, \ldots, \dim(\eta)$. This can be directly obtained from the algorithm described above. The cost of this algorithm is $\mathrm{O}(\xi q l)$ plus the cost of the original problem, where ξ is the dimension of the control vector; q is the number of random walks for each evaluation of inner product and l is the average length of the random walk, which is proportional to the number of time steps m of the original problem. When the dimension of the control vector is much smaller than the mesh size, and the required precision is relatively low (which allows us to choose a small q), the cost of solving the adjoint equation in this algorithm is a small overhead. This is particularly attractive especially compared to solving the exact solution of the unsteady adjoint equation, which is significantly more costly in computation time and storage than the original problem.

5. Analysis of the Monte Carlo method

In the last section, we derived the algorithm of using random walk Monte Carlo to solve the discrete adjoint equation, and theoretically proved that as the number of samples increases, the solution obtained by our method converges asymptotically to the exact solution. In practice, however, it is only possible to run a finite number of random walks with limited computational resources. In this section, we address the following questions: how much error is made by running a finite number of random walks, and how this error can be controlled and minimized.

Before we start, we definition the probable error in order to quantify the difference between the Monte Carlo approximation and the exact solution of the adjoint equation.

Definition 5.1. Let I be the value to be estimated by Monte Carlo method, and D be its unbiased estimator. The probable error for the Monte Carlo method is defined to be

$$r = \sup\left\{s : \mathcal{P}(|I - D| \geqslant s) > \frac{1}{2}\right\} \tag{20}$$

The probable error specifies the range which contains 50% of the possible values of the estimator. In the case of continuous distribution, this is equivalent to the definition in [9,25].

The probable error is closely related to the variance of the estimator. Suppose $D_1, \dots, D_q$ are independent and identically distributed samples of D, If the variance of estimator D is bounded, the Central Limit Theorem

$$\mathcal{P}\left(\frac{\sum D_i}{q} - I \leqslant x\right) \to \Phi(x)$$

holds. When q is large, the probable error of the average of the q samples is [9,27]

$$r \approx 0.6745\left(\frac{\operatorname{Var} D}{q}\right)^{1/2}. \tag{21}$$

Therefore, the probable error decreases when the number of samples q increases or when the variance of the estimator D decreases. Using this formula, we can estimate and control the probable error of our Monte Carlo method by estimating the variance of the D estimator.

In the rest of this section, we focus on the variance of the D estimator. To make mathematical derivation cleaner, we denote

$$p_{ij} = p(i,j)$$

to be the transition probability from state i to state j,

$$p_i = p(i, N+1)$$

to be the transition probability from state i to the final exit state, and

$$r_i = r(i)$$

to be the birth probability at state i.

5.1. Variance decomposition

Suppose the mean and variance of the D estimator of a random walk starting from state i are $\mathbf{E}_i(D)$ and $\mathbf{V}_i(D)$, respectively. Since the Markov chain at state i has p_i probability of going to the final exit state and p_{ij} probability of going to the jth state, we know

$$\mathbf{E}_i(D) = p_i\left(\frac{1}{p_i}b_i\right) + \sum_{j:A_{ij}\neq 0} p_{ij}\left(\frac{A_{ij}}{p_{ij}}\mathbf{E}_j(D)\right) = b_i + \sum_{j:A_{ij}\neq 0} A_{ij}\mathbf{E}_j(D),$$

which corresponds to the linear equation (7) we want to solve. Apply the same analysis to the expectation of D^2, we can obtain a formula for the variance of the D estimator,

$$\begin{aligned}\mathbf{V}_i(D) = \mathbf{E}_i(D^2) - \mathbf{E}_i(D)^2 &= p_i\left(\frac{b_i}{p_i}\right)^2 + \sum_{j:A_{ij}\neq 0} p_{ij}\left(\frac{A_{ij}^2}{p_{ij}^2}\mathbf{E}_j(D^2)\right) - \mathbf{E}_i(D)^2 \\ &= \left(\frac{b_i^2}{p_i} + \sum_{j:A_{ij}\neq 0}\frac{A_{ij}^2}{p_{ij}}\mathbf{E}_j(D)^2 - \mathbf{E}_i(D)^2\right) + \left(\sum_{j:A_{ij}\neq 0}\frac{A_{ij}^2}{p_{ij}}\mathbf{V}_j(D)\right).\end{aligned} \tag{22}$$

We can see that the variance of a random walk starting at state i comes from two parts, The first part

$$\mathbf{V}_i^{(1)}(D) = \frac{b_i^2}{p_i} + \sum_{j:A_{ij}\neq 0} \frac{A_{ij}^2}{p_{ij}} \mathbf{E}_j(D)^2 - \mathbf{E}_i(D)^2. \tag{23}$$

This part of variance is caused by the first step of the random walk. The second part

$$\mathbf{V}_i^{(2)}(D) = \sum_{j:A_{ij}\neq 0} \frac{A_{ij}^2}{p_{ij}} \mathbf{V}_j(D) \tag{24}$$

is a weighted average of the variance of random walks starting from all the states that state i leads to. This part of variance is caused solely by the random walk starting from the second step.

Similarly, we can calculate the variance of the D estimator of a random walk starting with birth probability r_i,

$$\begin{aligned}\mathbf{V}_r(D) = \mathbf{E}_r\left(D^2\right) - \mathbf{E}_r(D)^2 &= \sum_{i:r_i\neq 0} r_i \left(\frac{c_i^2}{r_i^2}\mathbf{E}_i\left(D^2\right)\right) - \mathbf{E}_r(D)^2 \\ &= \left(\sum_{i:r_i\neq 0} \frac{c_i^2}{r_i}\mathbf{E}_i(D)^2 - \mathbf{E}_r(D)^2\right) + \left(\sum_{i:r_i\neq 0} \frac{c_i^2}{r_i}\mathbf{V}_i(D^2)\right),\end{aligned} \tag{25}$$

where $\mathbf{E}_i(D)$ and $\mathbf{V}_i(D)$ are the mean and variance of the random walk starting deterministically from the state i. We can see that the variance of a random walk starting with birth probability r_i also comes from two parts. The first part

$$\mathbf{V}_r^{(1)}(D) = \sum_{i:r_i\neq 0} \frac{c_i^2}{r_i}\mathbf{E}_i(D)^2 - \mathbf{E}_r(D)^2 \tag{26}$$

is caused by the randomness of the birth state. The second part

$$\mathbf{V}_r^{(2)}(D) = \sum_{i:r_i\neq 0} \frac{c_i^2}{r_i}\mathbf{V}_i(D^2) \tag{27}$$

is a weighted average of the variance of random walks starting from all the possible birth states. This part of variance is caused by the random walk starting from the each possible birth states.

Based on this split of variance, the next section discusses choice of transition and birth probabilities p_i and r_i that reduces each component of the decomposition.

5.2. Choice of transition and birth probabilities

Finding the probabilities that make the variance as small as theoretically possible has been shown to be impractically time consuming for Monte Carlo linear solvers [6,8]. For this reason, we focus on finding 'almost optimal' transition and birth probabilities by minimizing upper bounds of the variances.

The upper bounds on which we base our almost optimal transition probabilities are

$$\mathbf{V}_i^{(1)}(D) = \frac{b_i^2}{p_i} + \left(\sum_{j:A_{ij}\neq 0} \frac{A_{ij}^2}{p_{ij}} \mathbf{E}_j(D)^2\right) - \mathbf{E}_i(D)^2 \leqslant \frac{b_i^2}{p_i} + \left(\sum_{j:A_{ij}\neq 0} \frac{A_{ij}^2}{p_{ij}}\right)\mathbf{B}^2 - \mathbf{E}_i(D)^2$$

and

$$\mathbf{V}_i^{(2)}(D) = \sum_{j:A_{ij}\neq 0} \frac{A_{ij}^2}{p_{ij}} \mathbf{V}_j(D) \leqslant \left(\sum_{j:A_{ij}\neq 0} \frac{A_{ij}^2}{p_{ij}}\right) \max_{j:A_{ij}\neq 0} \mathbf{V}_j(D),$$

where

$$\mathbf{B} = \max |\mathbf{E}_j(D)|$$

These bounds are chosen because individual $\mathbf{E}_j(D)^2$ and $\mathbf{V}_j(D)$ are not known a priori. They are therefore substituted by a common upper bound. The optimal p_{ij} and p_i for these upper bounds, under the constraints

$$\sum_j p_{ij} + p_i = 1 \quad \text{and} \quad p_{ij} \geqslant 0,$$

are given by the formulae[1]

$$p_{ij}^* = \frac{|A_{ij}|\mathbf{B}}{|b_i| + \sum_j |A_{ij}|\mathbf{B}} \tag{28}$$

and

$$p_i^* = \frac{|b_i|}{|b_i| + \sum_j |A_{ij}|\mathbf{B}}. \tag{29}$$

These are the almost optimal transition probabilities. Note that since $\mathbf{B}$ is not known a priori, it needs to be estimated unless $b_i = 0$.

Similarly, we construct upper bounds for $\mathbf{V}_r^{(1)}(D)$ and $\mathbf{V}_r^{(2)}(D)$:

$$\mathbf{V}_r^{(1)}(D) = \left(\sum_{i:r_i \neq 0} \frac{c_i^2}{r_i} \mathbf{E}_i(D)^2 \right) - \mathbf{E}_r(D)^2 \leqslant \left(\sum_{i:r_i \neq 0} \frac{c_i^2}{r_i} \right) \mathbf{B}^2 - \mathbf{E}_r(D)^2$$

$$\mathbf{V}_r^{(2)}(D) = \sum_{i:r_i \neq 0} \frac{c_i^2}{r_i} \mathbf{V}_i(D^2) \leqslant \left(\sum_{i:r_i \neq 0} \frac{c_i^2}{r_i} \right) \left(\max \mathbf{V}_i(D^2) \right)$$

The almost optimal birth probabilities that minimizes these upper bounds are given by the formula

$$r_i^* = \frac{|c_i|}{\sum_i |c_i|} \tag{30}$$

Because the almost optimal transition and birth probabilities minimize upper bounds of the estimator variance, we use this choice of probabilities in all our numerical experiments presented later in this paper.

5.3. Growth of variance

This section studies how the variance of our D estimator can grow assuming the almost optimal transition and birth probabilities derived in the last section. First, we plug the optimal probabilities into the upper bounds of $\mathbf{V}_i^{(1)}$ and $\mathbf{V}_i^{(2)}$, we get

$$\mathbf{V}_i^{(1)}(D) \leqslant (|b_i| + \mathbf{\Gamma}_i \mathbf{B})^2 - \mathbf{E}_i(D)^2$$

and

$$\mathbf{V}_i^{(2)}(D) \leqslant \left(\frac{|b_i|}{\mathbf{B}} + \mathbf{\Gamma}_i \right) \mathbf{\Gamma}_i \max_{j:A_{ij} \neq 0} \mathbf{V}_j(D),$$

where

$$\mathbf{\Gamma}_i = \sum_j |A_{ij}|.$$

The total variance is the sum of the two components of the variance decomposition, thus

$$\mathbf{V}_i(D) \leqslant \left((|b_i| + \mathbf{\Gamma}_i \mathbf{B})^2 - \mathbf{E}_i(D)^2 \right) + \left(\frac{|b_i|}{\mathbf{B}} \mathbf{\Gamma}_i + \mathbf{\Gamma}_i^2 \right) \max_{j:A_{ij} \neq 0} \mathbf{V}_j(D). \tag{31}$$

[1] (28) and (29) together minimizes upper bound of $\mathbf{V}_i^{(1)}(D)$; (28) alone minimizes upper bound of $\mathbf{V}_i^{(2)}(D)$ under the constraints.

This equation characterize the growth of variance as the Monte Carlo random walk proceeds. In the case of explicit time stepping, all j such that $A_{ij} \neq 0$ are in the next time step of i. As the random walk proceed through time steps, the variance can suffer from exponential growth if the multiplicative factor $(\frac{|b_i|}{\mathbf{B}}\mathbf{\Gamma}_i + \mathbf{\Gamma}_i^2)$ is greater than 1. If this is the case, the probable error can be too large for our Monte Carlo method to be practical. On the other hand, if this factor is less or equal to 1, the variance grow at most linearly. For this reason, the size of multiplicative factor is critical in the efficiency of our Monte Carlo method.

In the case of conservation law PDEs discretized with a positive coefficient scheme, we know the size of this multiplicative factor. The discrete conservation property guarantees that

$$\sum_j A_{ij} = 1,$$

for all but boundary grid points. Also, all A_{ij} are non-negative in a positive coefficient scheme. As a result of these two properties,

$$\Gamma_i = \sum_j |A_{ij}| = \sum_j A_{ij} = 1. \tag{32}$$

In addition, if the adjoint equation does not have a source term, then $b_i = 0$, and the multiplicative factor

$$\frac{|b_i|}{\mathbf{B}}\mathbf{\Gamma}_i + \mathbf{\Gamma}_i^2 = 1.$$

This implies that the probable error of our Monte Carlo method does not increase exponentially in this case. This is indeed true, as seen in our numerical experiments, that the error of our Monte Carlo adjoint solver decreases as the number of time steps gets larger. In case the adjoint equation have a source term, the discretized source term b_i is of order of Δt, thus the multiplicative factor

$$\frac{|b_i|}{\mathbf{B}}\mathbf{\Gamma}_i + \mathbf{\Gamma}_i^2 = 1 + \mathrm{O}(\Delta t).$$

This suggests that for a fixed T, as the discretization refines, the total amount of variance growth remain bounded.

For systems other than scalar transport equations, Eq. (32) may not be true. In many cases, a proper choice of preconditioner is required to prevent exponential growth of variance.

5.4. Choice of preconditioner

The choice of the preconditioner matrix P[2] in Eq. (6) controls the behaviors of the Neumann series, and influences the variance growth of the estimator D. A good choice can improve the precision of the result and reduce the cost by requiring fewer samples, while a bad choice can make the variance grow exponentially. Thus, the main purposes of the preconditioner is to control the multiplicative factor in the variance growth by reducing Γ_i. Just like choosing a preconditioner a linear system, there is no universal best choice. Still, there are a few preconditioners that we wish to mention in the setting of unsteady adjoint equation.

First of all, if the Jacobian matrix is diagonal dominant, a possible choice of preconditioner is the inverse of the diagonal part of the Jacobian. This method is called diagonal splitting. Diagonal splitting makes the Neumann series equivalent to the Jacobian iteration scheme, which has guaranteed convergence for diagonal dominant matrices.

Tan proposes a relaxed Monte Carlo linear solver in [9,27], which is equivalent to choosing a diagonal preconditioner that is not equal to the diagonal part of the Jacobian matrix. It was shown that this approach has improved performance over diagonal splitting for many problems.

Srinivasan [26] studied non-diagonal splitting Monte Carlo solvers, which is equivalent to choosing the preconditioner to be inverse of the diagonal and first sub-diagonal or super-diagonal of $\bar{A}$. His approach is suit-

[2] Not to be confused with the transition probability matrix $\mathbf{P}$.

able for a larger class of problems than diagonal conditioning. We have not yet investigated the last two more advanced preconditioner in the context of solving unsteady adjoint equations.

In the context of partial differential equations, choosing a preconditioner that transforms the random walk into the frequency space is an ideal currently being investigated. Preliminary results show that this preconditioner may be a solution to certain problems on which achieving a bounded variance growth using existing preconditioners is difficult.

6. Example of Monte Carlo algorithm: Burgers' equation

In this section, we demonstrate a concrete example of the Monte Carlo algorithm for solving adjoint equations. The example is Burgers' equation

$$\mathcal{R}(x,t) = u_t + \left(\frac{u^2}{2}\right)_x = 0,$$

discretized temporally by the forward Euler scheme, and spatially by the first order up-winding scheme. Our original problem is the fully discretized equation

$$\mathcal{R}_i^{(t)} = u_i^{(t)} - u_i^{(t-1)} + \frac{\Delta t}{\Delta x}\left(f_{i+\frac{1}{2}}^{(t-1)} - f_{i-\frac{1}{2}}^{(t-1)}\right) = 0, \quad t = 1,2,\ldots,m, \tag{33}$$

where f is the numerical flux computed using the up-winding formula

$$f_{i+\frac{1}{2}}^{(t)} = \begin{cases} \frac{1}{2}(u_i^{(t)})^2 & \text{if } u_i^{(t)} + u_{i+1}^{(t)} > 0 \\ \frac{1}{2}(u_{i+1}^{(t)})^2 & \text{if } u_i^{(t)} + u_{i+1}^{(t)} \leqslant 0. \end{cases}$$

In this case, the full state vector u is

$$u = (u_1^{(1)},\ldots,u_n^{(1)},\ u_1^{(2)},\ldots,u_n^{(2)},\ \ldots,\ u_1^{(m)},\ldots,u_n^{(m)})^{\mathrm{T}}$$

and the full constraint is

$$\mathcal{R} = (\mathcal{R}_1^{(1)},\ldots,\mathcal{R}_n^{(1)},\ \mathcal{R}_1^{(2)},\ldots,\mathcal{R}_n^{(2)},\ \ldots,\ \mathcal{R}_1^{(m)},\ldots,\mathcal{R}_n^{(m)})^{\mathrm{T}}$$

where n is the number of mesh points, and m is the number of time steps. The size of the adjoint equation as a linear system is $N = mn$.

For this example, we will analyze the structure of the adjoint equation, derive the birth probability matrix and transition probability matrix, and walk through the Monte Carlo algorithm. Let us begin with the constraint Jacobian matrix. The forward Euler temporal discretization scheme is a one-step scheme, and the constraint (33) at a time step t only depends on u at time step t and $t-1$. Hence, the only non-zero blocks in the Jacobian (4) are J_{tt} and $J_{tt-1},\ t = 1,\ldots,m$. Moreover, the equation is discretized using explicit scheme, so $\frac{\partial \mathcal{R}_i^{(t)}}{\partial u_j^{(t)}}$ is non-zero only if $i = j$. Therefore, the diagonal blocks of the Jacobian are identity matrices,

$$J_{tt} = \frac{\partial \mathcal{R}^{(t)}}{\partial u^{(t)}} = I.$$

In this case, we use no preconditioner, and

$$A = I - \left(\frac{\partial \mathcal{R}}{\partial u}\right)^{\mathrm{T}} = \begin{bmatrix} 0 & -J_{21}^{\mathrm{T}} & & \\ & 0 & \ddots & \\ & & \ddots & -J_{mm-1}^{\mathrm{T}} \\ & & & 0 \end{bmatrix}.$$

The Neumann series associated with this matrix has finite length m.

Each block J_{tt-1}^{T} in this matrix is well structured. We note that the residue (33) at mesh-point i only depends on u at mesh-point $i-1, x$ and $i+1$. Thus $\frac{\partial \mathcal{R}_i^{(t)}}{\partial u_j^{(t)}}$ is non-zero only if $i-1 \leqslant j \leqslant i+1$, and the off diagonal blocks J_{tt-1} are tri-diagonal matrices with entries

$$\begin{aligned} \frac{\partial \mathcal{R}_i^{(t)}}{\partial u_{i-1}^{(t-1)}} &= -a_i^{(t)} \frac{\Delta t}{\Delta x} u_{i-1}^{(t)}, \\ \frac{\partial \mathcal{R}_i^{(t)}}{\partial u_i^{(t-1)}} &= -1 - b_i^{(t)} \frac{\Delta t}{\Delta x} u_i^{(t)}, \\ \frac{\partial \mathcal{R}_i^{(t)}}{\partial u_{i+1}^{(t-1)}} &= -c_i^{(t)} \frac{\Delta t}{\Delta x} u_{i-1}^{(t)}, \end{aligned} \tag{34}$$

where

$$a_i^{(t)} = \begin{cases} -1 & \text{if } u_{i-1}^{(t)} + u_i^{(t)} > 0 \\ 0 & \text{if } u_{i-1}^{(t)} + u_i^{(t)} \leqslant 0 \end{cases}$$

$$b_i^{(t)} = \begin{cases} 1 & \text{if } u_{i-1}^{(t)} + u_i^{(t)} > 0 \text{ and } u_i^{(t)} + u_{i+1}^{(t)} > 0 \\ -1 & \text{if } u_{i-1}^{(t)} + u_i^{(t)} \leqslant 0 \text{ and } u_i^{(t)} + u_{i+1}^{(t)} \leqslant 0 \\ 0 & \text{otherwise} \end{cases}$$

$$c_i^{(t)} = \begin{cases} 1 & \text{if } u_i^{(t)} + u_{i+1}^{(t)} \leqslant 0 \\ 0 & \text{if } u_i^{(t)} + u_{i+1}^{(t)} > 0. \end{cases}$$

We compute the transition probabilities using (28), choosing the absorption probability to be 0 at $t < m$, and 1 at $t = m$. We get the transition probability matrix

$$\mathbf{P} = \begin{bmatrix} 0 & P_1 & & & 0 \\ & 0 & \ddots & & 0 \\ & & \ddots & P_{m-1} & 0 \\ & & & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

where the last row and column correspond to the final exit state, and each P_t is an n by n tri-diagonal matrix. We note that following this transition probability matrix, each step of a random walk goes one time step forward. At the next step, a random walk either stays at the same mesh-point, or goes to its left or right neighbor, with probabilities specified by P_t at the current time step. Therefore, in our Monte Carlo method, the spatial distribution of the random walks follows transition probabilities P_t at each time step, and all random walks are absorbed at the mth time step. This is consistent to the finite length of a Neumann series associated with the matrix A.

Assume that we are interested in solving for the adjoint solution at the first time step $\psi^{(1)}$, we can approximate it by starting q random walks from each of the n mesh-points at the first time step. Denote $\alpha[p,i]$ as the pth random walk starting at mesh-point i, and $\alpha_t[p,i]$ its position at time step t. The Monte Carlo algorithm in this case is

```
(1) For each $1 \leqslant p \leqslant q, 1 \leqslant i \leqslant n$, choose $\alpha_0[p,i] = x$ and initialize
    $W[p,i] = 1;\quad D[p,i] = b_i.$
    Then start from $t = 1$, do steps 2–4, until done with the last time step $t = m$.
(2) Solve the original problem at time step $t$ for $u^{(t)}$. Compute tri-diagonal matrix $-J_{t+1t}^{\mathrm{T}}$ using (34), and
    transition probabilities $P_t$ at this time step using (28).
```

(3) For each random walk, choose its next state $\alpha_{t+1}[p]$ according to P_t. Update
$$W[p,i] = W[p,i]w_{\alpha_t[p,i]\alpha_{t+1}[p,i]}; \quad D[p,i] = D[p,i] + W[p,i]b_{\alpha_{t+1}[p,i]}.$$
(4) Let $t = t+1$, if $t < m$, go to step 2.
(5) At last time step m, all random walks are absorbed. Compute the sample mean of the estimators $\frac{1}{q}\sum_{p=1}^{q} D[p,i]$ for each i, which is our approximation to $\psi_i^{(1)}$.

7. Numerical experiments

In this section, we use three numerical experiments to demonstrate the convergence property and scaling efficiency of our Monte Carlo adjoint solver as well as its performance in solving an inverse problem. These experiments are done with Burgers' equation, discretized with numerical scheme (33). Because we used an explicit temporal discretization scheme for the original problem, we use no preconditioning in the Monte Carlo algorithm (see Section 5.4). The transition probabilities and birth probabilities are chosen based on (28) and (30). The absorption probabilities are chosen to be 1 at the last time step, and 0 at other time steps except at boundaries. All experiments are done on a desktop with two Intel(R) Xeon(TM) 3.00 GHz CPUs, 2 GB memory, GNU/Linux 2.6.9-42.0.10.ELsmp.

7.1. Convergence of Monte Carlo adjoint solver

This experiment demonstrates that the Monte Carlo adjoint solution converges to the exact solution as the number of random walk increases. We solve the Burgers' equation with initial condition
$$u(x,0) = \sin(\pi x), \quad x \in [0,1] \tag{35}$$
in time interval $[0, 0.25]$. First-order up-winding finite volume scheme is used on 100 grid points uniformly space on $[0, 1]$; the CFL number for time integration is set to 0.5. On the other hand, the adjoint equation is solved with final condition
$$\phi(x, 0.25) = \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad x \in [0,1] \tag{36}$$
in the same time interval $[0, 0.25]$, where $\mu = 0.5$ and $\sigma = 0.2$. Both Burgers' equation and the adjoint equation use zero Dirichlet boundary condition.

Our Monte Carlo adjoint solver is used in four different settings. In each setting, the number of random walks starting from each grid point is, respectively, 1, 10, 100 and 1000. Griewank's checkpointing adjoint solver is used to compute the exact adjoint solution. Fig. 1 plots the Monte Carlo adjoint equation in each of the four cases on top of the exact solution. The Monte Carlo adjoint solution clearly converges towards the exact solution as the number of random walks increases.

The rate of this convergence is depicted in Fig. 2. The slope of the line in the log–log plot is roughly 0.5, indicating that our Monte Carlo adjoint solver converges at a rate of $\sqrt{q}$, which is the common rate of convergence in Monte Carlo methods.

7.2. Scaling efficiency

By avoiding trajectory storage and re-iteration, our Monte Carlo adjoint solver is especially competitive to exact solution methods when the number of time steps is large. This efficiency is demonstrated by this experiment.

In this experiment, the Burgers' equation is solved in time interval $[0, 0.25]$ with uniform grids of 10 different sizes, ranging from 10 to 10,000. The CFL number for time integration is set to 0.5, thus the number of time steps increases proportionally to the grid size. Similar to the first experiment, the initial condition for Burgers' equation is (35); the numerical scheme used is first-order up-winding finite volume. Again, we solve the adjoint equation in time interval $[0, 0.25]$ using both our Monte Carlo solver and Griewank's optimal checkpointing

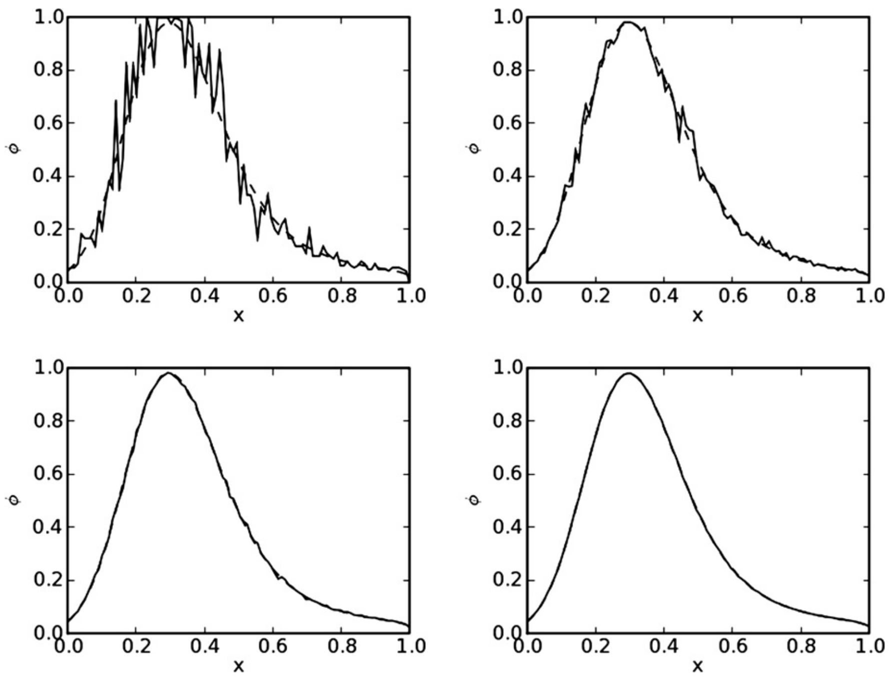

Fig. 1. Solutions of adjoint equation at time 0. Solid lines are solutions estimated by Monte Carlo method; dash lines are the exact solution. The number of random walks starting from each grid point is top-left plot: 1; top-right plot: 10; bottom-left plot: 100; bottom-right plot: 1000.

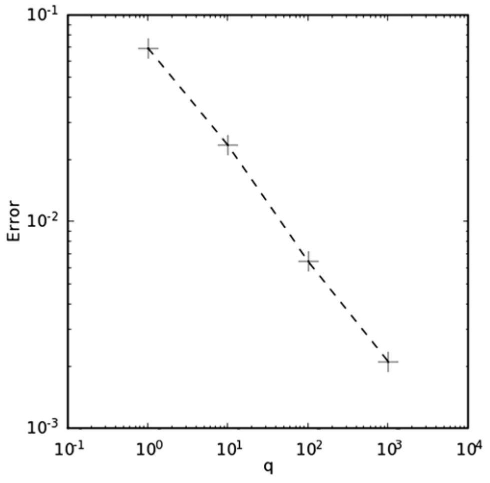

Fig. 2. Convergence of Monte Carlo adjoint solution. The horizontal axis is q, the number of random walks starting from each grid points; the vertical axis is the L^2 distance between Monte Carlo adjoint solution and the exact solution.

scheme. Two different settings, with $q = 1$ and $q = 10$, respectively, are used in the Monte Carlo solver. We then compare the time it takes using our Monte Carlo method in both settings to the time it takes using the optimal checkpointing scheme.

From the log–log plot in Fig. 3, we observe that the computation time of our Monte Carlo method in both settings is proportional to the square of grid size. This is because with fixed CFL number, the number of time steps grow linearly to the grid size, and the computational time is proportional to the product of the grid size and the number of time steps. In contrast, the computation time of the optimal checkpointing scheme grows faster, with a theoretical rate of $n^2 \log n$, where n is the grid size.

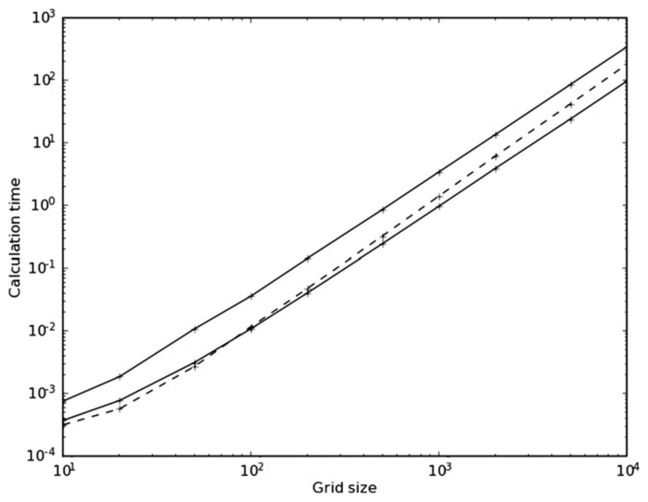

Fig. 3. The computation time of the adjoint solution as grid size increases. The dash line is the amount of time it takes to compute the exact adjoint solution using Griewank's optimal checkpointing scheme; the solid line above it is the amount time it takes to estimate the adjoint solution using Monte Carlo method using $q = 10$ random walks per grid point; the solid line below is the amount of time it takes to estimate the adjoint solution using Monte Carlo method using $q = 1$ random walks per grid point. The calculation is done on a desktop with two Intel(R) Xeon(TM) 3.00 GHz CPUs, 2GB memory, GNU/Linux 2.6.9-42.0.10.ELsmp. All calculations are single threaded.

This scaling efficiency of our Monte Carlo method is obtained without sacrificing the accuracy of its estimated solution. Fig. 4 shows the L^2 distance between the Monte Carlo adjoint solution and the exact solution for different grid sizes. As the grid size and number of time steps increases, the quality of the Monte Carlo adjoint solution increases for a fixed number of random walks per grid point. This result indicates that when the computation upscales as the spatial and temporal resolution increases, our Monte Carlo is more computationally efficient and produces more accurate estimated adjoint solution.

7.3. A Monte Carlo adjoint driven inverse problem

In this experiment, we test the performance of our Monte Carlo adjoint solver in solving a simple inverse problem: finding the initial condition of a Burgers' equation so that the solution at time $T = 0.25$ matches a prescribed target function

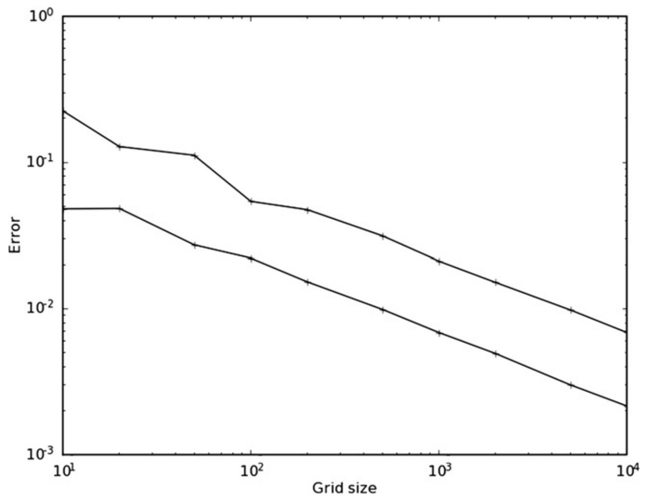

Fig. 4. L^2 estimation error of the Monte Carlo adjoint solver. The top line is the error for $q = 1$ random walks per grid point; the bottom line is the error for $q = 10$ random walks per grid point.

$$f_t(x) = \exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right), \quad x \in [0,1],$$

where $\mu = 0.5$ and $\sigma = 0.2$. The boundary condition is zero Dirichlet at both $x = 0$ and $x = 1$.

We solve the Burgers' equation using first-order up-winding finite volume scheme on 100 uniformly spaced grid points; the CFL number is set to 0.5. The adjoint solution is obtained by our Monte Carlo adjoint solver; the number of random walk starting from each grid point q is set to 1. This adjoint solution is used to drive a gradient based optimization procedure to minimizes the square L^2 difference between the solution at T and the prescribed function f_t. In this experiment, we use the BFGS quasi-Newton algorithm provided by Python module scipy.optimize. The initial guess fed into the optimization routine is f_t.

Due to insufficient precision of the gradient, calculated from the Monte Carlo adjoint solution, the optimization procedure terminated after 28 iterations without reaching its default error tolerance. The result of this optimization procedure and the corresponding solution at T is plotted in Fig. 5. For the purpose of comparison, a fully converged solution at iteration 68 driven by the exact adjoint solution is plotted in Fig. 6. Despite the wiggling fluctuations on the solution using Monte Carlo adjoint solver, it captures the shape of the solution. The corresponding Burgers' solution at T is also close to the target function. Moreover, the objective function, the square L^2 distance to the target function is 2.4×10^{-4}, more than 2 orders of magnitude smaller than that of the initial guess, which is 7.5×10^{-2}. This result indicates that the Monte Carlo adjoint solution, being an approximation itself, is useful in obtaining an approximate solution of optimization and inverse problems. The accuracy of the adjoint solution limits the accuracy of the computed gradient, preventing full convergence of gradient based optimization procedures.

Better convergence can be obtained if a smoothed version of the Monte Carlo adjoint solution is used to calculate the gradient. Fig. 7 shows the solution of the same inverse problem driven by Monte Carlo adjoint solutions smoothed by a Gaussian filter. The number of random walks starting from each grid point q is still set to 1. The σ of the Gaussian filter is 0.03. This time the optimization routine terminates after 58 iterations. Although the default error tolerance is still not reached, the quality of the solution is significantly improved. The final objective function, 3.2×10^{-5}, is also an order of magnitude smaller than the final objective function of the non-smoothed case.

This experiment concludes that despite being less accurate than the exact solution, the Monte Carlo adjoint is capable of reducing the objective function in optimization and inverse problems. In some practical problems where the Monte Carlo adjoint solver is more computationally efficient, it may be desirable to use the Monte

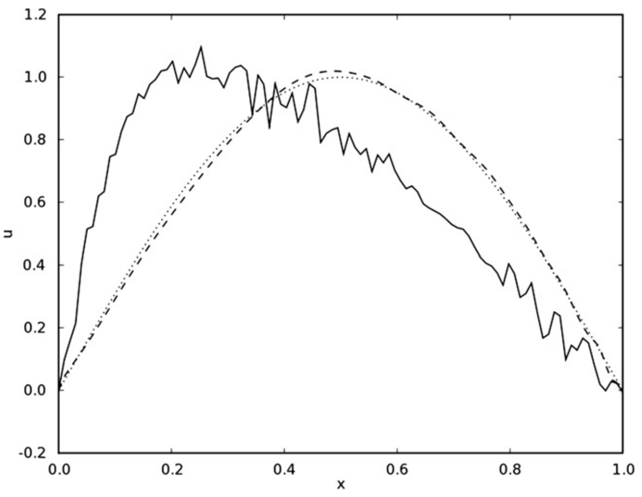

Fig. 5. Solving an inverse problem using Monte Carlo adjoint solver. The solid line is the solution to the inverse problem after 28 BFGS iterations. The dash line is the solution at time T of the Burgers' equation with the solid line as its initial condition. The dot line is the target function f_t. The dash line and dot line being close means that the solid line is an approximate solution to the inverse problem.

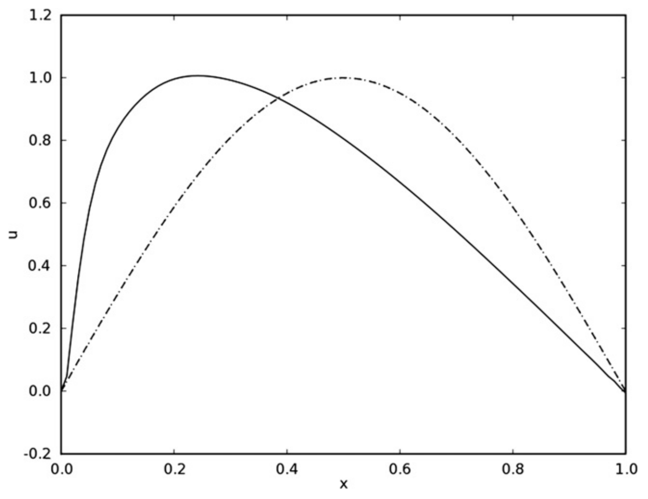

Fig. 6. Solving the same inverse problem using exact adjoint solution. The solid line is the solution to the inverse problem after 68 BFGS iterations (fully converged.) The dash line is the solution at time T of the Burgers' equation with the solid line as its initial condition. The dot line is the target function f_t. The dash line and dot line lying on top of each other means that the solid line is an accurate solution of the inverse problem.

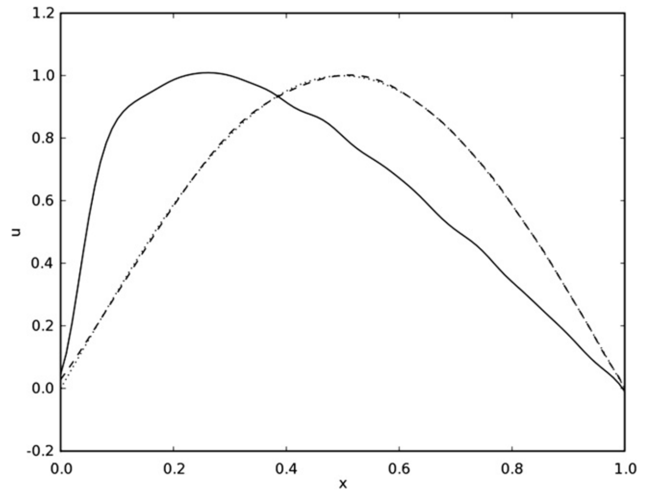

Fig. 7. Solving an inverse problem using smoothed Monte Carlo adjoint solver. The solid line is the solution to the inverse problem after 57 BFGS iterations. The dash line is the solution at time T of the Burgers' equation with the solid line as its initial condition. The dot line is the target function f_t. The dash line and dot line being almost on top of each other means that the solid line is an accurate approximate solution to the inverse problem.

Carlo adjoint solution to drive the optimization until further improvement is limited by its accuracy, then switch to an exact adjoint solver to ensure full convergence.

8. Future work

The $O(\sqrt{n})$ convergence rate of the Monte Carlo method (21) makes it unattractive when the variance of the estimator D is high. This is the case when we apply our method to vector transport equations such as Navier–Stokes equations. Therefore, our future work focuses on reducing the variance of our estimator.

(1) The variance of our estimator may be reduced by trying more advanced preconditioners [27,26], which can improve the convergence of the Neumann series. We are particular interested in investigating preconditioners in the case of vector partial differential equations, such as Navier–Stokes equation.
(2) We are interested in investigating Monte Carlo methods that are not based on the Neumann series. These methods are especially useful in case the Neumann series have poor convergence.

Our ultimate goal is to use this method in large simulations of physical systems, such as aerodynamic simulations, turbulence simulations and fluid-structure coupled simulations.

9. Conclusion

The Monte Carlo method is an efficient method to approximate the solution of the adjoint equation. The biggest advantages of this method are

(1) By only advancing forward in time, it avoids storing the full trajectory or iterating the original problem.
(2) It is easy to compute only part of the adjoint solution, or a linear function of the solution. This saves computation time in practice.
(3) It is conceptually easy to parallelize.

As we have demonstrated through our numerical experiments, our Monte Carlo method has better scaling efficiency than exact solution methods. By sacrificing some accuracy, choosing Monte Carlo adjoint solver in large scale calculations can be rewarding in terms of computation time and memory usage. This has been demonstrated in Burgers' equation. If this method can be efficiently generalized to vector transport equations, it will make very large scale calculation of their adjoint equations feasible.

Acknowledgment

This work is supported by US Department of Energy's ASC Program at Stanford.

References

[1] V.N. Alexandrov, Efficient parallel Monte Carlo methods for matrix computations, Mathematics and Computers in Simulation 47 (1998) 113–122.
[2] T. Bewley, P. Moin, R. Temam. Dns-based predictive control of turbulence: and optimal benchmark for feedback algorithms, Annual Research Briefs, Center of Turbulence Research, Stanford, 1993, pp. 3–14.
[3] A.E. Bryson, M.N. Desai, W.C. Hoffman, Energy state approximation in performance optimization of supersonic aircraft, Journal of Aircraft 6 (12) (1969) 488–490.
[4] I. Charpentier, Checkpointing schemes for adjoint codes: application to the meteorological model meso-nh, SIAM Journal on Scientific Computing 22 (6) (2001) 2135–2151.
[5] B. Choi, R. Temam, P. Moin, J. Kim, Feedback control for unsteady flow and its application to the stochastic burgers equation, Journal of Fluid Mechanics 253 (1993) 509–543.
[6] I. Dimov, Minimization of the probable error for some Monte Carlo methods, Mathematical Modelling and Scientific Computations, Publication House of the Bulgarian Acad. Sci., 1991, pp. 159–170.
[7] I. Dimov, Monte Carlo algorithms for linear problems, Lecture Notes of the 9th International Summer School on Probability Theory and Mathematical Statistics, 1998, pp. 51–71.
[8] I. Dimov, O. Tonev, Random walk on distant mesh points Monte Carlo methods, Journal of Statistical Physics 70 (1993) 1333–1342.
[9] C.J.K. Tan, et al., Relaxed Monte Carlo linear solver, in: Proceedings of the International Conference on Computational Sciences – Part I Table of Contents, 2001, pp. 1289–1298.
[10] S.E. Forsythe, R.A. Liebler, Matrix Inversion by a Monte Carlo Method, Mathematical Tables and Other Aids to Computation (1950) 127–129.
[11] M.B. Giles, N.A. Pierce, An introduction to the adjoint approach to design, Flow, Turbulence and Combustion 65 (2000) 393–415.
[12] M.B. Giles, N.A. Pierce, Adjoint error correction for integral outputs, Error Estimation and Adaptive Discretization Methods in Computational Fluid Dynamics, Springer-Verlag, 2002, pp. 47–96.
[13] M.B. Giles, E. Suli, Adjoint methods for pdes: a posteriori error analysis and postprocessing by duality, Acta Numerica 11 (2002) 145–236.

[14] A. Griewank, Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation, Optimization Methods and Software 1 (1992) 35–54.
[15] A. Griewank, A mathematical view of automatic differentiation, Acta Numerica (2003) 321–398.
[16] M.D. Gunzburger, S. Manservisi, The velocity tracking problem for Navier–Stokes flows with bounded distributed controls, SIAM Journal of Control and Optimization 37 (6) (1999) 1913–1945.
[17] M.D. Gunzburger, S. Manservisi, Analysis and approximation for linear feedback control for tracking the velocity in Navier–Stokes flows, Computer Methods in Applied Mechanics and Engineering 189 (3) (2000) 803–823.
[18] J. Hoffman, On duality based a posteriori error estimation in various norms and linear functionals for les, SIAM Journal on Scientific Computing 26 (1) (2004) 178–195.
[19] J. Hoffman, Weak uniqueness of the Navier–Stokes equations and adaptive turbulence simulation, entry for the Leslie Fox Prize, 2005.
[20] A. Jameson, Aerodynamic design via control theory, Journal of Scientific Computing 3 (1988) 233–260.
[21] N. Metropolisand, S. Ulam, The Monte Carlo method, Journal of the American Statistical Association 44 (1949) 335–341.
[22] G. Okten, Solving linear equations by Monte Carlo simulation, SIAM Journal of Scientific Computing 27 (2) (2005) 511–531.
[23] M. Pharr, P. Hanrahan, Monte carlo evaluation of non-linear scattering equations for subsurface reflection, in: Proceedings of SIGGRAPH, 2000.
[24] R.Y. Rubinstein, Simulation and the Monte Carlo Method, John Wiley and Sons, New York, 1981.
[25] I.M. Sobol, Monte Carlo Numerical Methods, Russian edition., Nauka, Moscow, 1973.
[26] A. Srinivasan, Improved Monte Carlo linear solvers through non-diagonal splitting, ICCSA (3) (2003) 168–177.
[27] C.J.K. Tan, Solving systems of linear equations with relaxed Monte Carlo method, The Journal of Supercomputing 22 (2002) 113–123.
[28] A. Walther, A. Griewank, Advantages of binomial checkpointing for memory-reduced adjoint calculations, Numerical Mathematics and Advanced Applications (2004) 834–843.
[29] J.R. Westlake, A Handbook of Numerical Matrix Inversion and Solution of Linear Equations, John Wiley and Sons, New York, 1968.