Algorithm 799: Revolve: An Implementation of Checkpointing for the Reverse or Adjoint Mode of Computational Differentiation

ANDREAS GRIEWANK and ANDREA WALTHER
Technical University Dresden

In its basic form, the reverse mode of computational differentiation yields the gradient of a scalar-valued function at a cost that is a small multiple of the computational work needed to evaluate the function itself. However, the corresponding memory requirement is proportional to the run-time of the evaluation program. Therefore, the practical applicability of the reverse mode in its original formulation is limited despite the availability of ever larger memory systems. This observation leads to the development of checkpointing schedules to reduce the storage requirements. This article presents the function `revolve`, which generates checkpointing schedules that are provably optimal with regard to a primary and a secondary criterion. This routine is intended to be used as an explicit "controller" for running a time-dependent applications program.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classifications—*Fortran 77*; G.1.4 [**Numerical Analysis**]: Quadrature and Numerical Differentiation—*Automatic differentiation*; G.1.6 [**Numerical Analysis**]: Optimization—*Gradient methods*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Dynamic programming*

General Terms: Algorithms

Additional Key Words and Phrases: Adjoint mode, checkpointing, computational differentiation, reverse mode

1. INTRODUCTION

The reverse mode of computational differentiation is a discrete analog of the adjoint method known from the calculus of variations [Griewank 2000]. The gradient of a scalar-valued function is yielded by the reverse mode (in its basic form) for no more than five times the operations count of

Authors' address: Institute of Scientific Computing, Technical University Dresden, Dresden, D-01062, Germany; email: griewank@math.tu-dresden.de; awalther@math.tu-dresden.de.

evaluating the function itself. This bound is completely independent of the number of independent variables. More generally, this mode allows the computation of Jacobians for at most five times the number of dependents times the effort of evaluating the underlying vector function. However, the spatial complexity of the basic reverse mode, i.e., its memory requirement, is proportional to the temporal complexity of the evaluation of the function itself, since all intermediate results need to be recorded. Therefore, the practical use of the good temporal complexity result for the reverse mode seems to be severely limited by the amount of memory required.

The article is organized as follows. After the completion of this introduction, Section 2 explains the theoretical background. Furthermore, it contains two optimality statements and their proofs. The function `revolve` and auxiliary routines are detailed in Section 3. In Section 4, an example illustrates the application of `revolve`. We present some numerical results achieved on the example of the preceding section in Section 5. Some conclusions are drawn in Section 6.

1.1 A Simple Example

The desire to keep storage requirements for the reverse mode under control leads to checkpointing schedules in which memory usage is traded for run-time by reversing only parts of the program at a time. The following example will illustrate the idea in principle.

Consider the composite $y = g(f(x))$ of two vector functions $g, f : \mathbb{R}^n \to \mathbb{R}^n$ and the following procedure for evaluating it:

—Evaluate $z = f(x)$

—Evaluate $y = g(z)$.

The basic reverse mode would now evaluate first $z = f(x)$ and $y = g(z)$, recording all intermediate results generated during the calculation of these two functions. On the basis of these execution logs one may then perform a single reverse sweep

—Evaluate $\bar{z} = \bar{y}^T g'(z)$

—Evaluate $\bar{x} = \bar{z}^T f'(x)$

yielding the corresponding adjoint $\bar{x} = \bar{y}^T \partial y / \partial x$. Here, $\bar{y}$ is a row vector of constant weights that linearly determines the gradient $\bar{x}$ at the given point x.

First let us consider the situation where f and g are elemental operations like sin() or exp(). Then the computed values that are needed on the way back in order to reverse the function evaluation can be easily recorded on the way forward. When a program execution contains millions or billions of individual operations, this recording of all intermediate values calculated is no longer a trivial matter. Therefore, suppose that the evaluation of f and g involves many intermediate quantities which cannot all be recorded and kept in storage simultaneously. Then we may use the initial state x as a checkpoint to reduce the memory requirement according to the following scheme:

—Save x on a stack.

—Evaluate $z = f(x)$ without recording any intermediate results.

—Evaluate $y = g(z)$, recording all intermediate results.

—Compute $\bar{z} = \bar{y}^T g'(z)$ by a reverse sweep on g.

—Restore x from the stack.

—Reevaluate $z = f(x)$, recording all intermediate results.

—Compute $\bar{x} = \bar{z}^T f'(x)$ by a reverse sweep on f.

The key feature of this scheme is a trade-off between spatial and temporal complexity. Now we only have to store x rather than all intermediates involved in evaluating f, which presumably requires much less storage. However, we pay for this storage reduction with an increase in the operations count, as f needs to be evaluated again after g has been reversed so that the recording of its intermediates can be freed up. This technique can be applied recursively to a nested structure of subroutine calls or loops, which may involve any number of checkpoints during the overall function evaluation. Under somewhat idealized assumptions it was shown in Griewank [1992] that the trade-off between temporal and spatial complexity can be chosen such that both grow logarithmically relative to the time required to evaluate the function itself. Grimm et al. [1996] proved that this scheme is in fact optimal. A practical implementation faces two basic challenges.

The first task is the selection of checkpoints. Here, the question is at which points of the whole computational process one should place the checkpoints to achieve an optimal reduction of the storage requirement. In the example above, there is only one checkpoint at the initial state. But is this optimal? Perhaps it would be better to select another time during the evaluation of f or g as checkpoint and/or to choose more than one checkpoint. The second task is to manage all the information at every checkpoint that is necessary. This requires saving the system state at the checkpoint and restoring it in order to repeat the next computational steps, when that becomes necessary. Also, the values of the adjoints must be managed to perform the successive reverse sweeps. In the example above, one has to store the value of x at the beginning of the computation and the result $\bar{z}$ of the first part of the reverse sweep. For recomputing f one needs to assign the stored value of x to the current value of x. Finally, one has to perform the second part of the reverse sweep starting from the stored results of the first part.

1.2 The Definition of Time Steps

One may distinguish between explicitly time-dependent problems and general stepwise evaluation procedures. Explicitly time-dependent problems perform a sequence of transformations with a counter that can be thought of as discrete time. In other words, an iteration calculates the propagation of a state vector from one time step to the next. Typical examples are discretizations of continuous time-evolutions modeled by ordinary or partial differential equations.

Formally, any function evaluation procedure in an imperative programming language can be interpreted as a time-dependent problem with each single assignment being viewed as a transformation on the state space represented by the allocated program variables. For a single-assignment code the statement counter would then represent the discrete pseudotime. For the purpose of checkpointing, this interpretation is not very useful because each individual transformation has an extremely small operations count compared to the size of the state space. However, by simply combining several thousand successively executed statements into one computational step, one may obtain a more appropriate ratio between its operations count and the size of the state space. To distinguish it from the individual arithmetic operations and elementary function evaluations, we will refer to such an aggregate computation as a time step.

Generally, the time steps should be chosen such that the length of the recording needed to perform the corresponding reverse steps is at least as large as the size of the state space. We will assume here that the user can break the program down appropriately, for example at subroutine calls. Then he or she may use our software tool `revolve` to perform adjoint calculation with checkpointing.

2. MINIMIZING FORWARD STEPS AND SET CHECKPOINTS

In this section, we consider checkpointing schedules that were called divide-and-conquer strategies in Grimm et al. [1996]. After a short theoretical description, we prove two propositions of optimality.

2.1 The Underlying Theory

The function `revolve` implements a refined version of the algorithm described in Griewank [1992]. This algorithm addresses the first challenge mentioned in Section 1, namely that of selecting the checkpoints. Logarithmic growth of the spatial complexity in reverse mode can be achieved by letting `revolve` choose the right checkpoints. The routine `revolve` sets the checkpoints in a binomial fashion, and the intermediate values are mostly recalculated instead of being recorded. To apply this algorithm, the user must be able to break the function evaluation into a sequence of time steps that are roughly of equal size in terms of computational work.

The name `revolve` was chosen to reflect the fact that a reversal of an evolution is performed. The checkpoints can be thought of as pointers representing the intermediate states of the evolution. The application of the checkpointing schedule realized with `revolve` performs the following do-loop (e.g., Griewank [1992] and Grimm et al. [1996]) in order to reverse the function evaluation:

```
0: Initialization: Reserve space for $s$ checkpoints and set the first one to the
   initial state.
   do end = final, 2, -1
    1: Multiple forward: Starting from the last checkpoint assigned, advance to
       step end-1, performing forward time steps without recording of interme-
       diates. If one or more checkpoints are free, set as many of them as
       possible to intermediate states along the way.
    2: Combined reverse: Perform a forward time step with recording of inter-
       mediates to step end and perform a reverse sweep to step end-1 to
       calculate the adjoints. If step end-1 is a checkpoint, free it for subsequent
       use.
   end do
```

The phrase "set as many checkpoints to intermediate states as possible" in action 1 refers to the minimum of the number of free checkpoints and the number of states between the current and the penultimate state. Usually the second restriction only applies during the very end of the calculation when most checkpoints have been freed up. Earlier on there will often be no checkpoint available at all. When there is, the question is of course where the checkpoint(s) should be set. One fairly obvious strategy would be an equal spacing among the intermediate states between the current and the penultimate state. This would work quite well but is not optimal in that it may require more repeated forward time steps than actually needed for the reversal of the given number of time steps.

Let $\beta(s, t)$ be the maximal length, i.e., the maximal number of time steps, of any computational chain that can be reversed by the procedure described above with a maximum of s checkpoints and a maximum of t forward steps from any one of the states. Then it is shown in Griewank [1992] that

$$\beta(s, t) = \binom{s + t}{s}. \tag{1}$$

With this equality one obtains a logarithmic dependence of the memory requirement and of the number of operations relative on the run-time of the function evaluation [Griewank 1992]. Also it is possible to calculate the value of the third quantity if one has the value of two of the three quantities: maximal number of steps, maximal number of checkpoints, and maximal number of repeated forward steps without recording of intermediates.

For example, if there are 10 time steps and the possibility to store three checkpoints, then one gets $t = 2$ according to equality (1). Figure 1 depicts the procedure described above for these values of s, t, and the given number of time steps. The value of $\beta(s, t - 1)$ (here $\beta(3, 1) = 4$) determines the next checkpoint [Griewank 1992] after the initial one at zero.

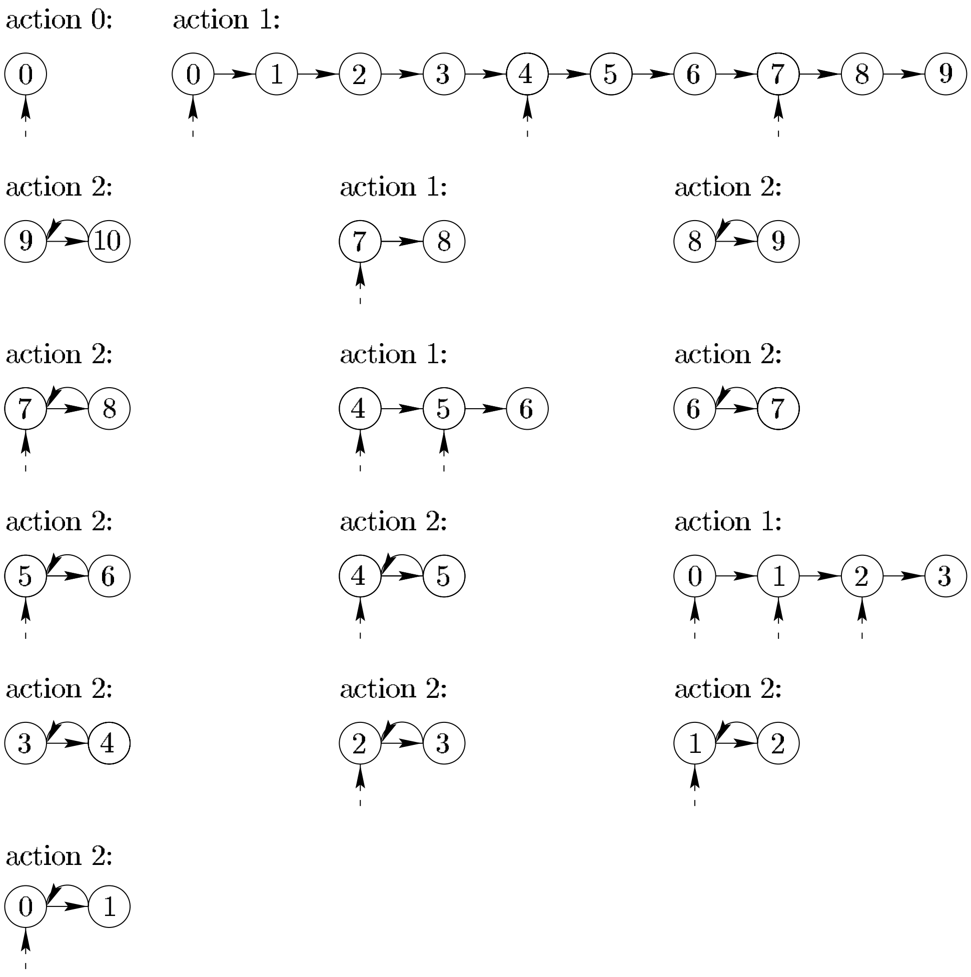

Fig. 1. Evaluation process using `revolve`.

Therefore state 4 is also marked as checkpointed. Finally the execution of action 1 marks state 7 as checkpoint, because $\beta(s - 1, t - 1)$ defines the number of steps from the second to the third checkpoint, and $\beta(2, 1)$ is equal to 3. The next subsection contains further theoretical results.

2.2 Some Complexity Results

There are at least three questions to be answered. The first one is how often we have to restore data from the checkpoints to advance to the penultimate state. Also, one wishes to know how many forward steps are necessary to reverse a whole chain with a given number of time steps. Here, only forward steps without recording of intermediates are counted because each forward step is recorded exactly once in connection with a reverse step. Therefore, the number of forward steps with recording of intermediates is given by the number of reverse steps. Finally, we wish to determine the minimal number of times a current state is saved onto the stack of checkpoints.

The following observation suffices to answer the first question. If m denotes the total number of time steps to be reversed, m reverse steps are necessary. Before each combined reverse step, excluding the first one, the forward sweep starts at the currently last checkpoint and ends at the penultimate state. Therefore, it is necessary to restore the data of the last checkpoint. Hence, data from a checkpoint are read $m - 1$ times. The following proposition gives the answer to the second question.

Proposition 1. *Let $p(m, s)$ denote the minimal number of extra forward steps that are needed to reverse a sequence of m time steps storing up to s checkpoints at any time. Then $p(m, s)$ must satisfy*

$$p(m, s) = \begin{cases} m(m-1)/2 & \text{if } s = 1 \\ \min\limits_{1 \le \tilde{m} < m} \{\tilde{m} + p(\tilde{m}, s) + p(m - \tilde{m}, s - 1)\} & \text{if } s > 1 \end{cases} \tag{2}$$

and takes the explicit form

$$p(m, s) = tm - \beta(s + 1, t - 1), \tag{3}$$

where t is the unique integer satisfying $\beta(s, t - 1) < m \le \beta(s, t)$.

Proof. For reversing $m > 1$ time steps, the first checkpoint is set to the initial state and the second checkpoint to a state $\tilde{m}$ with $1 \le \tilde{m} < m$. Then we have to reverse $m - \tilde{m}$ time steps on the right of the second checkpoint with up to $s - 1$ checkpoints set at any time and $\tilde{m}$ time steps on the left of the second checkpoint with a maximum of s checkpoints at any time. Also, we need $\tilde{m}$ forward steps to reach the state $\tilde{m}$. Hence, $p(m, s)$ must satisfy (2). A detailed proof of Eq. (2) can be found in Walther [1999]. Now identity (3) is shown by induction.

Induction ($s = 1$). For $s = 1$, one has

$$\beta(1, t - 1) = t < m \le t + 1 = \beta(1, t) \Leftrightarrow t = m - 1.$$

The checkpoint must be set to the initial state, and for the first reverse step one needs a minimum of t forward steps. To perform the next reverse step, $t - 1$ forward steps are necessary and so on. Therefore, the minimal number of forward steps is equal to

$$t + \cdots + 1 = \frac{1}{2}(t^2 + t) = t(t + 1) - \binom{t + 1}{2} = p(t + 1, 1) = p(m, 1).$$

Induction ($m = 1$). It is clear that no forward step without recording of intermediates is necessary, because one recorded forward step and one reverse step suffice to reverse a single time step. Also, for $t = 0$

$$\beta(s, -1) = 0 < 1 = m = \beta(s, 0),$$

so that

$$p(1, s) = 0 \cdot 1 - \beta(s + 1, -1) = -\binom{s}{s+1} = 0$$

as asserted.

Induction Step in Lexicographical Order of (s, m). The numbers $s > 1$ and $m > 1$ are given. It is obvious that they determine a unique $t \in \mathbb{N}$ with

$$\beta(s, t - 1) < m \leq \beta(s, t).$$

Assume that the assertion is true for all $(\tilde{s}, \tilde{m})$ with $\tilde{s} < s$ or $\tilde{s} = s$ and $\tilde{m} < m$. Now it will be shown that equality (3) is valid if the second checkpoint is set to a state satisfying the two conditions (4) and (5) below. Then it will be proved that more than $tm - \beta(s + 1, t - 1)$ forward steps are necessary if the second checkpoint is set to a state not satisfying these two conditions.

Let the second checkpoint be set to a state $\hat{m}$ satisfying

$$\beta(s, t - 2) \leq \hat{m} \leq \beta(s, t - 1) \tag{4}$$

and

$$\beta(s - 1, t - 1) \leq m - \hat{m} \leq \beta(s - 1, t). \tag{5}$$

One obtains from (4) and (5) that

$$\max\{\beta(s, t - 2), m - \beta(s - 1, t)\} \leq \hat{m} \leq \min\{\beta(s, t - 1), m - \beta(s - 1, t - 1)\}.$$

This valid domain for $\hat{m}$ is shaded gray in Figure 2. As one can see from this figure there is usually a range of choices $\hat{m}$. Now it will be shown that then $tm - \beta(s + 1, t - 1)$ forward steps are necessary for reversing m time steps.

It follows from (4) and the induction hypothesis that

$$(t - 1)\hat{m} - \beta(s + 1, t - 2)$$

equals the minimal number of forward steps for reversing $\hat{m}$ time steps. Similarly one obtains from (5) and the induction hypothesis that

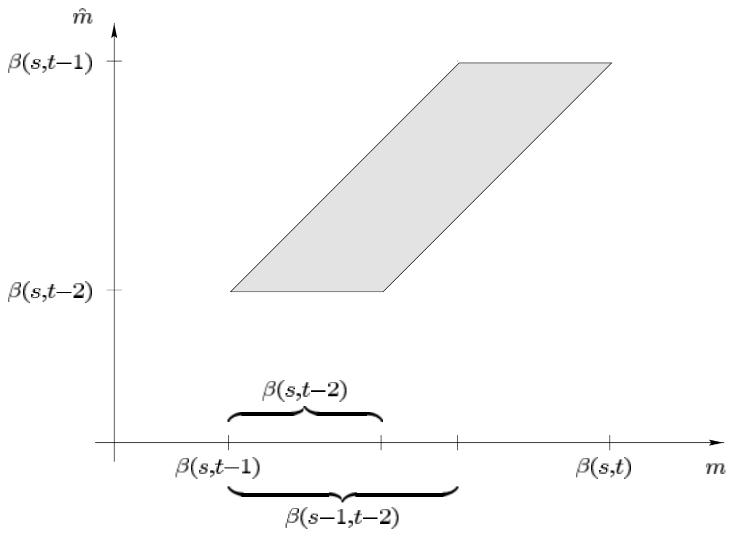

Fig. 2. Valid domain for the optimal value $\hat{m}$ of $\tilde{m}$.

$$t(m - \hat{m}) - \beta(s, t - 1)$$

forward steps are necessary for reversing $m - \hat{m}$ time steps. Finally, one needs $\hat{m}$ forward steps to go to $\hat{m}$.

Hence, if the second checkpoint is set to a state $\hat{m}$ satisfying (4) and (5), the minimal number of forward steps for reversing m time steps is given by

$$\hat{m} + (t - 1)\hat{m} - \beta(s + 1, t - 2) + t(m - \hat{m}) - \beta(s, t - 1) =$$

$$= tm - \beta(s + 1, t - 1) = p(m, s).$$

It remains to be shown that the number of forward steps for reversing m time steps becomes greater than $tm - \beta(s + 1, t - 1)$ if the second checkpoint is set to a state $\tilde{m}$ not satisfying (4) and (5). Note that $p(\tilde{m}, s)$ and $p(m - \tilde{m}, s - 1)$ are convex functions in $\tilde{m} \in \{1, \ldots, m-1\}$. Therefore,

$$p(\tilde{m}, s) + p(m - \tilde{m}, s - 1) + \tilde{m}$$

is also convex in $\tilde{m}$ with $\tilde{m} \in \{1, \ldots, m-1\}$. A convex function can have at most one interval where the function is minimal. For that reason it is sufficient to show that the minimal number of forward steps becomes greater than $tm - \beta(s + 1, t - 1)$ if

$$\tilde{m} = \max\{\beta(s, t - 2), m - \beta(s - 1, t)\} - 1$$

or

$$\tilde{m} = \min\{\beta(s, t-1), m - \beta(s-1, t-1)\}+1.$$

First, the case $\tilde{m} = \min\{\beta(s, t-1), m - \beta(s-1, t-1)\}+1$ will be examined. One possibility is that $\beta(s, t-1) < m - \beta(s-1, t-1)$. Then one obtains

$$\beta(s, t-1) < \tilde{m} = \beta(s, t-1)+1 < m \leq \beta(s, t)$$

as well as $\beta(s-1, t-1) \leq m - \tilde{m} < \beta(s-1, t)$. Therefore, we need a minimum of $t\tilde{m} - \beta(s+1, t-1)$ forward steps for reversing $\tilde{m}$ time steps. Also, one can conclude that $t(m - \tilde{m}) - \beta(s, t-1)$ forward steps are necessary to reverse $m - \tilde{m}$ time steps. Consequently, if the second checkpoint is set to state $\tilde{m} = \beta(s, t-1)+1$ the number of forward steps for reversing m time steps is given by

$$\tilde{m} + t\tilde{m} - \beta(s+1, t-1) + t(m - \tilde{m}) - \beta(s, t-1).$$

It follows that

$$p(m, s) - \tilde{m} - t\tilde{m} + \beta(s+1, t-1) - t(m - \tilde{m}) + \beta(s, t-1) = -1.$$

Hence, more than $p(m, s)$ forward steps are needed if the second checkpoint is set to the state $\tilde{m} = \beta(s, t-1)+1$. Now consider the second possibility, namely $\beta(s, t-1) = m - \beta(s-1, t-1)$. Then once again

$$\beta(s, t-1) < \tilde{m} = \beta(s, t-1)+1 < m \leq \beta(s, t).$$

Also, $\beta(s-1, t-2) \leq m - \tilde{m} = \beta(s-1, t-1)-1 < \beta(s-1, t-1)$ is valid because $s > 1$ and $t > 0$. From the induction hypothesis, we obtain that a total of $t\tilde{m} - \beta(s+1, t-1)$ forward steps are necessary to reverse the states located on the left of the second checkpoint and that $(t-1)(m - \tilde{m}) - \beta(s, t-2)$ forward steps to reverse $m - \tilde{m}$ time steps on the right of the second checkpoint. This yields

$$p(m, s) - \tilde{m} - t\tilde{m} + \beta(s+1, t-1) - (t-1)(m - \tilde{m}) + \beta(s, t-2) = -2.$$

Therefore, we need more than $p(m, s)$ forward steps to reverse m time steps if the second checkpoint is set to state $\beta(s, t-1)+1$. At last, let $\beta(s, t-1)$ be greater than $m - \beta(s-1, t-1)$. It follows that $\tilde{m} = m - \beta(s-1, t-1)+1$ as well as

$$\beta(s, t-2) < \tilde{m} < \beta(s, t-1)$$

and

$$\beta(s-1, t-2) \leq m - \tilde{m} = \beta(s-1, t-1)-1 < \beta(s-1, t-1)$$

because $s > 1$ and $t > 0$. Now we can conclude from the induction hypothesis that the minimal number of forward steps for reversing $\tilde{m}$ time steps is given by $(t-1)\tilde{m} - \beta(s+1, t-2)$. Also, $(t-1)(m-\tilde{m}) - \beta(s, t-2)$ forward steps are needed to reverse $m - \tilde{m}$ time steps. Finally, one obtains

$$p(m, s) - \tilde{m} - (t-1)\tilde{m} + \beta(s+1, t-2) - (t-1)(m-\tilde{m}) + \beta(s, t-2) = -1.$$

Now it is shown that more than $p(m, s)$ forward steps are needed to reverse m time steps if the second checkpoint is set to state

$$\tilde{m} = \min\{\beta(s, t-1), m - \beta(s-1, t-1)\} + 1.$$

Second, consider the case $\tilde{m} = \max\{\beta(s, t-2), m - \beta(s-1, t)\} - 1$. Let $\beta(s, t-2)$ be smaller than $m - \beta(s-1, t)$. It follows that

$$\beta(s, t-2) \le \tilde{m} = m - \beta(s-1, t) - 1 < \beta(s, t-1)$$

and

$$\beta(s-1, t) < m - \tilde{m} \le \beta(s-1, t+1).$$

Using the induction hypothesis we obtain that $(t-1)\tilde{m} - \beta(s+1, t-2)$ forward steps are necessary to reverse the left-hand side of the second checkpoint and $(t+1)(m-\tilde{m}) - \beta(s, t)$ forward steps are necessary to reverse the right-hand side of the second checkpoint. It follows that

$$p(m, s) - \tilde{m} - (t-1)\tilde{m} + \beta(s+1, t-2) - (t+1)(m-\tilde{m}) + \beta(s, t) = -1.$$

Therefore, more than $p(m, s)$ forward steps have to be performed to reverse m time steps if state $\tilde{m} = m - \beta(s-1, t) - 1$ is chosen as the second checkpoint. Now assume that

$$\beta(s, t-2) = m - \beta(s-1, t).$$

Then one can conclude from $\tilde{m} = \beta(s, t-2) - 1 \ge 1$ that $t > 2$ as well as

$$\beta(s, t-3) \le \tilde{m} < \beta(s, t-2)$$

and

$$\beta(s-1, t) < m - \tilde{m} \le \beta(s-1, t+1).$$

Hence, one needs at least $(t-2)\tilde{m} - \beta(s+1, t-3)$ forward steps to reverse $\tilde{m}$ time steps and at least $(t-1)(m-\tilde{m}) - \beta(s, t)$ forward steps to reverse $m - \tilde{m}$ time steps. This yields

$$p(m, s) - \tilde{m} - (t-2)\tilde{m} + \beta(s+1, t-3) - (t+1)(m-\tilde{m}) + \beta(s, t) < -2.$$

Consequently, more than $p(m, s)$ forward steps are performed to reverse m time steps if the second checkpoint is set to the state $\beta(s, t - 2) - 1$. Finally, the case $\beta(s, t - 2) > m - \beta(s - 1, t)$ will be examined. One has $\tilde{m} = \beta(s, t - 2) - 1 \geq 1$. It follows that $t > 2$ as well as

$$\beta(s, t - 3) \leq \tilde{m} < \beta(s, t - 2)$$

and

$$\beta(s - 1, t - 1) < m - \tilde{m} \leq \beta(s - 1, t).$$

From the induction hypothesis we obtain that at least $(t - 2)\tilde{m} - \beta(s + 1, t - 3)$ forward steps are needed to reverse $\tilde{m}$ time steps and at least $t(m - \tilde{m}) - \beta(s, t - 1)$ forward steps to reverse $m - \tilde{m}$ time steps. For that reason

$$p(m, s) - \tilde{m} - (t - 2)\tilde{m} + \beta(s + 1, t - 3) - t(m - \tilde{m}) + \beta(s, t - 1) = -1.$$

Hence, identity (3) is proved because the minimal number of forward steps for reversing m time steps is also greater than $p(m, s)$ if the second checkpoint is set to a state $\tilde{m} = \max\{\beta(s, t - 2), m - \beta(s - 1, t)\} - 1$. □

Our primary objective was the minimization of the total number of forward steps. As a secondary objective we consider now the minimization of the total number of times a current state is saved onto the stack of checkpoints. This criterion further reduces the range of possible $\hat{m}$ without narrowing it down to a single value, as stated in the following result.

PROPOSITION 2. *Among all checkpoint schedules satisfying Proposition 1 the minimal number of times a current state is saved onto the stack of checkpoints is given by*

$$q(m, s) = \begin{cases} \beta(s - 1, t - 1) & \text{if } m \leq \beta(s, t - 1) + \beta(s - 1, t - 1) \\ m - \beta(s, t - 1) & \text{if } m \geq \beta(s, t - 1) + \beta(s - 1, t - 1) \end{cases}$$

where again $\beta(s, t - 1) < m \leq \beta(s, t)$.

PROOF. The function $q(m, s)$ is well defined because $\beta(s - 1, t - 1) = m - \beta(s, t - 1)$ if m equals $\beta(s, t - 1) + \beta(s - 1, t - 1)$. To prove the assertion an induction argument is used once again:

Induction ($s = 1$). The checkpoint is set only once to the initial state. Therefore the number of times a current state is saved onto the stack of checkpoints is equal to 1. Also, one has

$$\beta(1, t - 1) = t < m \leq t + 1 = \beta(1, t - 1) + \beta(0, t - 1) \Leftrightarrow t = m - 1$$

and hence $q(m, 1) = \beta(0, m - 2) = 1$.

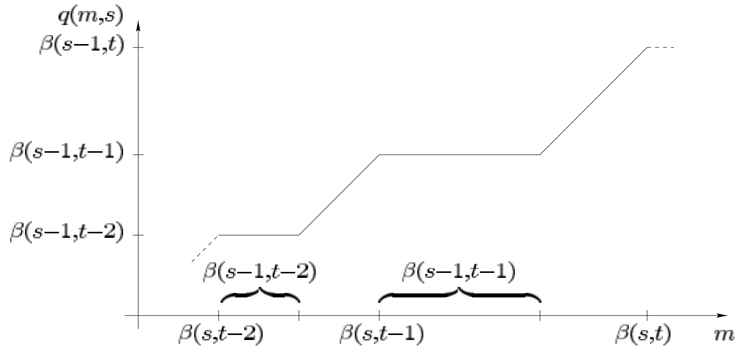

Fig. 3. The function $q(m, s)$.

Induction ($m \in \{1, 2\}$). Only one checkpoint is set to the initial state. In the case $m = 1$, a forward step with recording of intermediates and a reverse step will be performed. If $m = 2$ we advance with a forward step without recording of intermediates to state 1. Then two combined forward and reverse steps are necessary. Hence, for $m \in \{1, 2\}$ the number of times a current state is saved onto the stack of checkpoints is equal to 1. Also, the inequalities

$$\beta(s, -1) + \beta(s - 1, -1) = 0 \le 1 = m \le 1 = \beta(s, 0)$$

and

$$\beta(s, 0) + \beta(s - 1, 0) = 2 \le 2 = m \le s + 1 = \beta(s, 1)$$

hold for $m = 1$ and $m = 2$, respectively. This yields $q(1, s) = 1 - \beta(s, -1) = 1$ if $m = 1$ and $q(2, s) = 2 - \beta(s, 0) = 1$ for the case $m = 2$.

Induction Step in Lexicographical Order of (s, m). The given numbers $s > 1$ and $m > 2$ uniquely determine $t \in \mathbf{N}_0$ with $t > 0$ and $\beta(s, t - 1) < m \le \beta(s, t)$. Let the assertion be true for all $(\tilde{s}, \tilde{m})$ with $\tilde{s} < s$ or $\tilde{s} = s$ and $\tilde{m} < m$.

First, assume that m is not greater than $\beta(s, t - 1) + \beta(s - 1, t - 1)$. Now set the second checkpoint to a state $\hat{m}$ satisfying

$$\beta(s, t - 2) \le \hat{m} \le \beta(s, t - 2) + \beta(s - 1, t - 2) \tag{6}$$

and

$$\beta(s - 1, t - 1) \le m - \hat{m} \le \beta(s - 1, t - 1) + \beta(s - 2, t - 1). \tag{7}$$

One can prove that a corresponding $\hat{m}$ exists. The domain of possible $\hat{m}$ is illustrated by Figure 4. It follows from Eq. (6) and the induction hypothesis that $\beta(s - 1, t - 2)$ checkpoints need to be set to reverse $\hat{m}$ time steps. Also, one obtains from Eq. (7) and the induction hypothesis that for

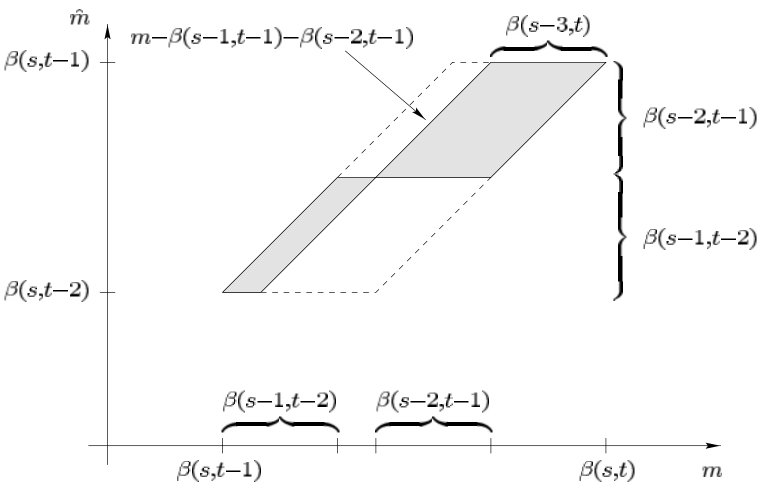

Fig. 4. The optimal domain of $\hat{m}$ for $m > 2$.

reversing $m - \hat{m}$ time steps the minimal number of times a current state is saved onto the stack of checkpoints equals $\beta(s - 2, t - 1)$. Therefore, the minimal number of checkpoints that have to be set is given by

$$\beta(s - 1, t - 2) + \beta(s - 2, t - 1) = \beta(s - 1, t - 1) = q(m, s)$$

if $\hat{m}$ satisfies (6) and (7).

It remains to be shown that if the second checkpoint is set to a state $\tilde{m}$ not satisfying (6) and (7) a current state must be saved more than $\beta(s - 1, t - 1)$ times onto the stack of checkpoints. One has to examine three possibilities.

At first the case $\tilde{m}$ satisfies (6), and

$$\beta(s - 1, t - 1) + \beta(s - 2, t - 1) < m - \tilde{m} < \beta(s - 1, t)$$

will be studied. Again, $\beta(s - 1, t - 2)$ checkpoints need to be set to reverse $\tilde{m}$ time steps, and $m - \tilde{m} - \beta(s - 1, t - 1)$ checkpoints need to be set to reverse $m - \tilde{m}$ time steps. Therefore,

$$\beta(s - 1, t - 2) + m - \tilde{m} - \beta(s - 1, t - 1) > \beta(s - 1, t - 1)$$

equal the minimal number of times a current state is saved onto the stack of checkpoints.

Now consider the case $\tilde{m}$ satisfies (7) and

$$\beta(s, t - 2) + \beta(s - 1, t - 2) < \tilde{m} \leq \beta(s, t - 1).$$

It follows from the induction hypothesis that $\tilde{m} - \beta(s, t - 2)$ checkpoints must be set to reverse the left-hand side of the second checkpoint, and

$\beta(s-2, t-1)$ checkpoints must be set to reverse the right-hand side of the second checkpoint. Hence, a current state must be saved at least

$$\tilde{m} - \beta(s, t-2) + \beta(s-2, t-1) > \beta(s-1, t-1)$$

times onto the stack of checkpoints.

If $\tilde{m}$ satisfies neither (6) nor (7) then one has

$$m = \tilde{m} + m - \tilde{m} > \beta(s, t-1) + \beta(s-1, t-1).$$

This is a contradiction to the assumption that $m \leq \beta(s, t-1) + \beta(s-1, t-1)$.

Consequently, it has been proved that $q(m, s)$ denotes the minimal number of times a current state must be saved onto the stack of checkpoints if the number of forward steps is also optimized and the inequality

$$\beta(s, t-1) \leq m \leq \beta(s, t-1) + \beta(s-1, t-1)$$

is valid. The value $q(m, s)$ is reached if the second checkpoint is set to a state $\hat{m}$ that satisfies (6) and (7).

Now assume that

$$\beta(s, t-1) + \beta(s-1, t-1) \leq m \leq \beta(s, t), \tag{8}$$

and set the second checkpoint to a state $\hat{m}$ satisfying

$$\beta(s, t-2) + \beta(s-1, t-2) \leq \hat{m} \leq \beta(s, t-1) \tag{9}$$

and

$$\beta(s-1, t-1) + \beta(s-2, t-1) \leq m - \hat{m} \leq \beta(s-1, t). \tag{10}$$

Again one proves easily that this choice of $\hat{m}$ is possible. We can conclude from (9), (10), and the induction hypothesis that

$$q(\hat{m}, s) = \hat{m} + \beta(s, t-2)$$

as well as $q(m - \hat{m}, s-1) = m - \hat{m} - \beta(s-1, t-1)$. Therefore, at least

$$q(\hat{m}, s) + q(m - \hat{m}, s) = m - \beta(s, t-1) = q(m, s)$$

checkpoints must be set to reverse m time steps if $\hat{m}$ satisfies (9) and (10).

It remains to be proved that a current state must be saved more than $q(m, s)$ times onto the stack of checkpoints, if a state $\tilde{m}$ not satisfying (9) and (10) is chosen as second checkpoint. Again one has to examine three possibilities.

First, consider the case where $\beta(s, t-2) \leq \tilde{m} < \beta(s, t-2) + \beta(s-1, t-2)$ and inequality (10) holds. Then, one obtains from the induction hypothesis

$$q(\tilde{m}, s) = \beta(s-1, t-2)$$

and $q(m-\tilde{m}, s-1) = m - \tilde{m} - \beta(s-1, t-1)$. It follows that the minimal number of times a current state must be saved onto the stack of checkpoints equals

$$q(\tilde{m}, s) + q(m-\tilde{m}, s-1) > m - \beta(s, t-1) = q(m, s).$$

Second, assume that $\tilde{m}$ satisfies (9) and

$$\beta(s-1, t-1) \leq m - \hat{m} < \beta(s-1, t-1) + \beta(s-2, t-1).$$

It follows from the induction hypothesis that $q(\tilde{m}, s) = \tilde{m} - \beta(s, t-2)$ and $q(m-\tilde{m}, s) = \beta(s-2, t-1)$. This yields that

$$q(\tilde{m}, s) + q(m-\tilde{m}, s) > m - \beta(s, t-1) = q(m, s)$$

checkpoints have to be set.

Finally the case where $\tilde{m}$ satisfies neither (9) nor (10) will be considered. One obtains

$$m = \tilde{m} + m - \tilde{m} < \beta(s, t-1) + \beta(s-1, t-1)$$

and therefore a contradiction to assumption (8).

Now it has been shown that for $m \geq \beta(s, t-1) + \beta(s-1, t-1)$, the minimal number of times a current state is saved onto the stack of checkpoints is given by $q(m, s)$. This optimal value is reached if the second checkpoint is set to a state $\hat{m}$ satisfying (9) and (10). Hence, the assertion has been established. □

Figure 3 shows the function $q(m, s)$. As can be seen it is piecewise linear with the slope alternating between 0 and 1.

In Figure 4, the two gray parallelograms mark the valid domain of $\hat{m}$ so that the number of forward steps without recording of intermediates and the number of times a current state is saved onto the stack of checkpoints are minimal. It is obvious that there are still some degrees of freedom. For the implementation of the algorithm presented above, namely the function `revolve`, the state $\hat{m}$ is selected as follows:

$$\hat{m} = \begin{cases} \beta(s, t-2) & \text{if } m \leq \beta(s, t-1) + \beta(s-2, t-1) \\ \beta(s, t-1) & \text{if } m \geq \beta(s, t) - \beta(s-3, t) \\ m - \beta(s-1, t-1) - \beta(s-2, t-1) & \text{otherwise} \end{cases}$$

with $\beta(\tilde{s}, t) \equiv 0$ if $\tilde{s} < 0$. Because of the proofs above, this choice guarantees that `revolve` will need the minimal number of forward steps and the minimal number of times a current state must be saved onto the stack of checkpoints.

After the derivation of the theoretical results, the application of `revolve` will be described.

3. USING REVOLVE

This section presents the use of `revolve` as an explicit "controller" for running time-dependent applications in the reverse mode of computational differentiation with checkpointing. Here, the attribute "time-dependent" merely requires that the program can be broken down into a sequence of steps as discussed in Section 1. Both C and Fortran 77 versions of `revolve` are provided.

Since `revolve` involves only a few integer operations, its run-time is truly negligible within any nontrivial application. To apply `revolve` the user must have procedures for the following tasks:

(a) Advancing the modeled system to a certain state

(b) Saving the current state onto a stack of checkpoints

(c) Restoring the most recently saved program state

(d) Performing one combined forward and adjoint step (with special provisions for the very first time this happens)

Through an encoding of its return value, `revolve` asks the calling program to perform one of these actions, which are referred to as follows:

(a) ADVANCE

(b) TAKESHOT

(c) RESTORE

(d) YOUTURN (FIRSTURN)

The C version of `revolve` has a return value of the enumeration type

```
enum action { advance, takeshot, restore, firsturn, youturn,
              terminate, error};
```

and the following calling sequence

```
enum action revolve(int* check,int* capo,int* fine,
                    int snaps, int* info)
```

We will refer to the first three arguments of `revolve` as the *schedule state*. The last parameter `info` controls the volume of information about actions performed and contains information about the kind of error in the case of an irregular termination.

On the first call to `revolve` the parameter `check` must be set to -1 so that appropriate initializations can be performed internally. Later `check` represents the number of checkpoints stored on the stack at any one time. The beginning of the subrange currently being processed is defined by the state number capo. The value of the state number fine determines the end of the subrange currently being processed. So the pair (capo, fine) always represents the initial and final state of the subsequence of states currently being traversed backward.

In contrast to the first three parameters, snaps is called by value and remains unaltered during the call. The variable snaps contains the upper bound on the number of checkpoints stored at any time and is selected by the user, possibly with the help of the routine expense described below. It is assumed that snaps is never greater than 64; otherwise the execution is stopped. If even more checkpoints are needed, the user must change the source code. For each interval actually under consideration, revolve calculates an appropriate t called reps to determine the state to which the next checkpoint will be set, if that is necessary. Once again, the upper bound of reps equals 64. If the calculated value of reps exceeds 64, an error message is given, the execution is stopped, and the user has to change the code. The upper bounds are almost always adequate. For example, 184,756 time steps can be reversed with 10 checkpoints, i.e., snaps $=$ 10, and $t = 10$, i.e., reps $=$ 10.

The following conditions are necessary and sufficient for a regular response of revolve:

$$\texttt{check} \geq -1 \text{ and } \texttt{capo} \leq \texttt{fine}.$$

If either condition is violated, revolve assigns an appropriate value to info and returns error. If check $= -1$ and capo $=$ fine, nothing is done internally, and terminate is recommended as action, i.e., returned as value. This situation occurs when the last reverse step has been taken, and it represents the only fixed point in the sense that the other parameters do not matter and remain unchanged, as well.

The last parameter info determines how much information about the actions performed will be printed. When info $= 0$, no information is sent to standard output. When info > 0, the first call of revolve produces an output that contains a prediction of the number of forward steps and of the factor by which the execution will be slower than a regular forward simulation. Furthermore, before terminate is returned, the total counts for the number of forward steps, savings of current states onto the stack, and actions taken are printed on standard output. If an error occurs the return value of info contains information about the reason for the irregular termination of revolve.

To choose an appropriate value of snaps, a function to estimate the run-time factor incurred by revolve for a particular value of snaps is available. It is called expense and has the syntax

```
    double expense(int steps, int snaps);
```

The ratio $p($`steps`, `snaps`$)$`steps` with $p(.,\ .)$ as defined in Proposition 1 is returned. The ratio corresponds to the run-time factor of the execution relative to the run-time of one forward step.

Furthermore, we provide the auxiliary function `adjust` with the syntax

```
int adjust(int steps);
```

This function can be used to determine a value of `snaps` so that the increase in spatial complexity is approximately equal to the increase in temporal complexity. For this purpose, `adjust` computes a return value satisfying `snaps` $\approx \log_4($`steps`$)$ based on the theory developed in Griewank [1992].

The use of `revolve` is illustrated in the following code segment from an actual program.

The problem-dependent aspects are incorporated into the definitions of the three procedures `forward(u)`, `forwardrec(u)`, and `reverse(bu,bz)`, which also communicate with each other through global data structures. Here, `u`, `bu`, and `bz` are real vectors describing states and their adjoints. An instantiation in the case of Burgers' equation will be given in the next section.

```
. . .  /* declarations and initializations */
      /* initialize schedule state: */
check = –1; capo = 0; fine = steps;
      /* initialize schedule parameter: */
snaps = adjust(steps);
do {
  oldcapo = capo;
  whatodo = revolve(&check, &capo, &fine, snaps, &info);
  switch(whatodo)
    {/* Advancing the system to state capo: */
    case advance: for(j=oldcapo;j  < capo;j++)
                  forward(u);
                  break;
      /* Saving the current state as checkpoint: */
    case takeshot: for(i=0;i<n;i++)
                   ustor[check][i]= u[i];
                   break;
      /* Advance with recording and perform first reverse step */
    case firsturn: forwardrec(u);
                 for(i=0;i<n;i++) /* Initialize adjoints */
                   {bu[i] = u[i]-ustar[i]); bz[i] = 0;}
                 reverse(bu,bz); /* First reverse step */
                 break;
      /* Subsequent, combined forward/reverse steps */
    case youturn: forwardrec(u);
                  reverse(bu,bz);
                  break;
    case restore: for(i=0;i<n;i++)
                  u[i]=ustor[check][i];
    case error:   printf(" \n scheduling error \n ");
                  exit(-1);
    }
  }while (whatodo != terminate);
```

After this introduction to the use of `revolve`, some further technical explanations seem appropriate.

During the forward sweep the user is free to change the parameters `fine` and `snaps` from call to call, except that `fine` may never be less than the current value of `capo`. These new values are taken into account immediately for determining the place of the next checkpoint. Such adjustments may be needed when the total number of time steps to be taken is not known a priori. One then has two possibilities. First, one should choose the initial value of `fine` − `capo` as an upper bound of the real number of time steps. When the number of time steps becomes known exactly the user can then adjust `fine` accordingly. The reduction is possible without any problems but naturally does not lead to an optimal checkpoint schedule because the determination of the checkpoint placements is not optimal. If it turns out that the chosen initial value of `fine` is not sufficient, the user is allowed to increase `fine`. This requires the provision of additional checkpoints by increasing `snaps` if the number of checkpoints already set is equal to the current value of `snaps`. However, then the efficiency is usually somewhat reduced compared to a situation where `steps` = `fine` − `capo` is known correctly a priori.

If `snaps` is changed and the new value is smaller than the number of checkpoints actually set, an error message occurs.

The relation `fine` = `capo`+1 initiates the reverse sweep, which happens automatically if `fine` is left constant, as `capo` is eventually moved up to it by repeated calls to `revolve`. Once the first reverse or restore action has been taken, only the last parameter should be changed.

4. APPLICATION TO BURGERS' EQUATION

The partial differential equation

$$u_t(x,t) + u(x,t)u_x(x,t) - \nu u_{xx}(x,t) = z(x)u(x,t)$$

$$x \in (0,1), \quad t > 0, \quad \nu > 0,$$

with the boundary and initial conditions

$$u(0,t) = \alpha, \quad u(1,t) = \beta, \quad u(x,0) = g(x)$$

is a version of Burgers' equation. The mathematical setting of this equation is well known (e.g., Lions [1971]).

The nonlinear term uu_x is responsible for shocks that, depending on the initial state $g(x)$, may cause waves to break at a critical time $t_*(g)$ if the diffusion term νu_{xx} vanishes exactly. When ν is small, a rather sharp transition layer emerges as time tends to infinity. The right-hand side may be thought of as a control with the coefficient function $z(x)$ being constant in time, just like the boundary conditions.

Without the diffusion term the time-evolution of $u(x, t)$ would be reversible in that $\tilde{u}(x, t) \equiv u(1 - x, - t)$ solves the same partial differential equation after interchanging the two boundary condition values and switching the sign of $z(x)$. In rewriting the full equation in terms of $\tilde{u}$, the diffusion term also changes sign and thus makes the inverse problem extremely ill conditioned. In other words, the state u at some time t_0 cannot be reconstructed from the state at some later time $t_1 > t_0$ with any accuracy at all. Here, this means that the computational process cannot be reversed without the use of checkpoints or a complete execution log.

4.1 Stationary Solutions

Especially when $\alpha < \beta$ and/or when the coefficient function $z(x)$ is with a large absolute value negative, there is a good chance of a smooth equilibrium

$$u_*(x) = \lim_{t \to \infty} u(x, t)$$

being reached. Assuming that $u_*(x)$ is smooth, we find by substituting it into Burgers' equation with the time-derivative u_t vanishing exactly that

$$-\nu u''_*(x) + u_*(x)u'_*(x) = z(x)u_*(x) \quad \text{with} \quad u_*(0) = \alpha \quad \text{and} \quad u_*(1) = \beta.$$

In other words, we have a nonlinear two-point boundary value problem in ordinary differential equations. Such problems are in general quite hard to solve, especially when there are sharp transition layers in the interior or on the boundary. Hence it makes sense to solve the time-dependent problem approximately and hope for convergence to the correct limit.

4.2 The Continuous Adjoint for a Least-Squares Fit

Now suppose we have for some "large" time T a desired profile $u_*(x)$ that is to be obtained approximately by adjusting the control function $z(x)$. More specifically, we wish to minimize the functional

$$\varphi(z) \equiv \frac{1}{2}\int_0^1 (u(x, T; z) - u_*(x))^2 dx$$

where $u(x, t; z)$ is the solution of Burgers' equation with the given $z(x)$. To this end we need the gradient of $\varphi(z)$ or that of an appropriate discretization. In the function space, the gradient can be expressed in terms of the so-called costate $\bar{u}(x, t; z)$ which is defined as the solution of the adjoint differential equation

$$\bar{u}_t(x, t; z) + u(x, t; z)\bar{u}_x(x, t; z) + \nu\bar{u}_{xx}(x, t; z) = -z(x)\bar{u}(x, t; z),$$

$$x \in (0, 1), \quad 0 < t < T$$

with the boundary and terminal conditions

$$\bar{u}(0, t\,; z) = 0 = \bar{u}(1, t\,; z), \quad \bar{u}(x, T\,; z) = u(x, T\,; z) - u_*(x).$$

Note that $\bar{u}(x, t\,; z)$ is subject to a terminal condition at $t = T$ and must therefore be integrated backward in time. The forward trajectory $u(x, t\,; z)$ occurs as a coefficient function and must therefore be known before the costate-equation can be integrated. During the backward integration the gradient $\bar{z}(x) = \partial\varphi/(\partial z(x))$ can be accumulated as the integral

$$\bar{z}(x) = \int_T^0 u(x, t\,; z)\bar{u}(x, t\,; z)(-dt).$$

The practical use of the adjoint equation requires its discretization. Here we use a conservative central difference discretization that happens to commute with adjoining [LeVeque 1992]. Therefore the gradient $\bar{z}(x)$ obtained by adjoining the discretized time evolution is identical to the solution of the discretized adjoint equation.

4.3 A Central Difference Space Discretization

On a uniform grid $x_i = ih$ for $i = 0, \ldots, n$ with $h = 1/n$ we approximate

$$u_i(t) \approx u(x_i, t), \qquad \frac{u_{i+1}(t) - 2u_i(t) + u_{i-1}(t)}{h^2} \approx u_{xx}(x_i, t)$$

and

$$\frac{u_{i+1}^2(t) - u_{i-1}^2(t)}{4h} \approx u(x_i, t)u_x(x_i, t) \equiv \frac{1}{2}\frac{\partial}{\partial x}u^2(x, t)\big|_{x=x_i}.$$

Now we fix a certain $t \in (0, T)$ and throughout use the abbreviations

$$u_i = u_i(t), \quad u = (u_0, \ldots, u_n), \quad \bar{u}_i = u_i(t), \quad \text{and } \bar{u} = (\bar{u}_0, \ldots, \bar{u}_n).$$

Substituting the approximations u_i for $i = 1, \ldots, n-1$ into Burgers' equation and observing the boundary conditions $u_0(t) = \alpha$, $u_n(t) = \beta$ we obtain for $u_i' \equiv (\partial u_i(t))/\partial t$ the system of $n-1$ ordinary differential equations

$$u_i' = F_i(u)$$

with

$$F_i(u) \equiv z_i u_i - \frac{u_{i+1}^2 - u_{i-1}^2}{4h} + \nu \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}.$$

From the corresponding space discretization for the adjoint differential equation it follows similarly that

$$\bar{u}_i' = \bar{F}_i(u, \bar{u})$$

with

$$\bar{F}_i(\mathbf{u}, \bar{\mathbf{u}}) \equiv - z_i\, \bar{u}_i - \frac{u_i(\bar{u}_{i+1} - \bar{u}_{i-1})}{2h} - \nu \frac{\bar{u}_{i+1} - 2\bar{u}_i + \bar{u}_{i-1}}{h^2}$$

for $i = 1, \ldots, n - 1$. Define the auxiliary functions

$$F_0(\mathbf{u}) = -\frac{u_1^2}{4h} + \frac{\nu}{h^2} u_1$$

and

$$F_n(\mathbf{u}) = \frac{u_{n-1}^2}{4h} + \frac{\nu}{h^2} u_{n-1}.$$

Then one can check that the right-hand side vectors

$$F(\mathbf{u}) \equiv (F_i(\mathbf{u}))_{0 \le i \le n}$$

and

$$\bar{F}(\mathbf{u}, \mathbf{u}) \equiv (\bar{F}_i(\mathbf{u}, \mathbf{u}))_{0 < i < n}$$

are adjoint to each other in that for all vectors $\mathbf{u}$, $\bar{\mathbf{u}}$

$$\bar{F}(\mathbf{u}, \bar{\mathbf{u}}) \equiv -[D\tilde{F}(\mathbf{u})]^T \bar{\mathbf{u}}$$

with

$$D\tilde{F}(\mathbf{u}) \equiv \left(\frac{\partial F_i}{\partial u_1}, \ldots, \frac{\partial F_i}{\partial u_{n-1}}\right)_{0 \le i \le n}.$$

Interestingly this adjointness relation does not hold when the quadratic term uu_x is discretized as $u_i(u_{i+1} - u_{i-1})/(h/2)$ rather than by the conservative scheme described above. The following C functions integrate these space discretizations by the forward Euler scheme and can be used in the program fragment given above.

```
forward()
  { int i, double dui = 0;
    for(i=1;i<n;i++)
    { uux = (u[i+1]*u[i+1]-u[i-1]*u[i-1])/dx/4;
      uxx = (u[i+1]-2*u[i]+u[i-1])/dx/dx;
      u[i-1]+=dui;
      dui = dt*(z[i]*u[i] - uux + nu*uxx); }
    u[n-1] += dui; }
forwardrec()
  { for(int i=0;i<n;i++)
      rec[i] = u[i];
    forward(); }
reverse()
  { int i; double bui=0;
    for(i=1;i<n;i++)
    { bz[i] += dt*bu[i]*rec[i];
      uux = rec[i]*(bu[i+1]-bu[i-1])/dx/2;
      uxx = (bu[i+1]-2*bu[i]+bu[i-1])/dx/dx;
      bu[i-1] += bui;
      bui = dt*(z[i]*bu[i] + uux + nu*uxx); }
    bu[n-1] += bui; }
```

The update for the adjoints `bz[i]` $\approx z(x_i)$ in `reverse` were derived directly from the discrete forward routine. These basic discretization schemes were used and listed because of their simplicity. Similar results with regard to computational complexity were obtained with more sophisticated schemes [Osher and Solomon 1982] that yield qualitatively better results in the transition layers.

5. NUMERICAL RESULTS

Finally, we present some numerical results that were obtained using `revolve`.

Figure 5 shows the discrete state vector `u` at the iterations 500, 1000, 5000, 10000, 20,000, and 40,000. The initial state is the straight line running through the boundary values $u_0 = \alpha = 2/3$ and $u_n = \beta = -1/3$. The "control" function on the right-hand side was selected as $z(x) = 1 - 2e^{-x}$ for no particular reason. The target profile $u_*(x)$ was set piecewise constant to the boundary values with a jump at the midpoint $x = 0.5$. Because the diffusion coefficient $\nu = 0.001$ was selected to be quite small, the solution develops a rather sharp transition layer just below $x = 0.6$.

Figure 6 displays the gradient `bz` of the functional φ evaluated at the states drawn in Figure 5. For this purpose the forward integration was interrupted and the adjoint equation integrated backward. As is to be expected from the theory, the gradients converge at about the same linear rate as the iterates themselves. The kink at the midpoint originates from the jump in the target profile $u_*(x)$.

Figure 7 displays the run-time for the adjoint calculation with `revolve` as a function of the ratio between the overall number of time steps (here 5000) and the maximal number of checkpoints specified as a percentage.

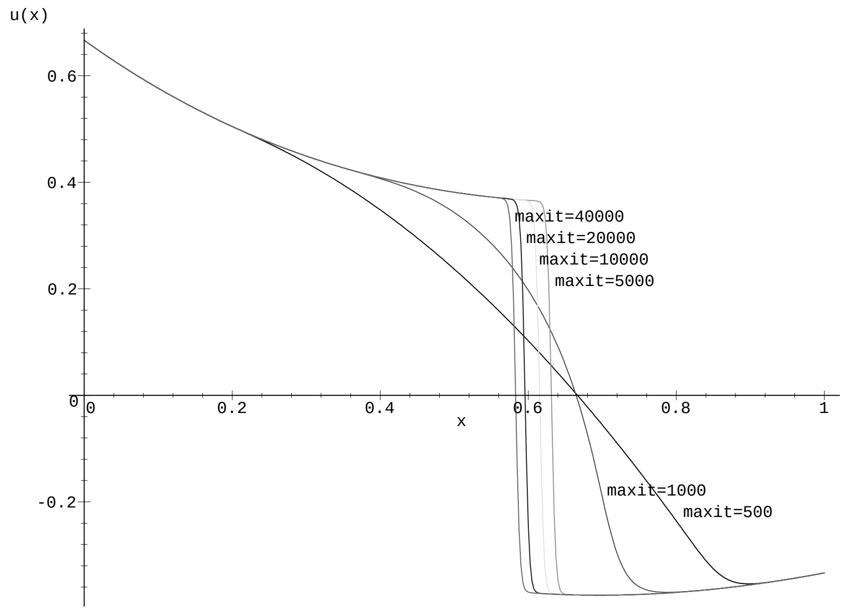

Fig. 5. Development of the function u.

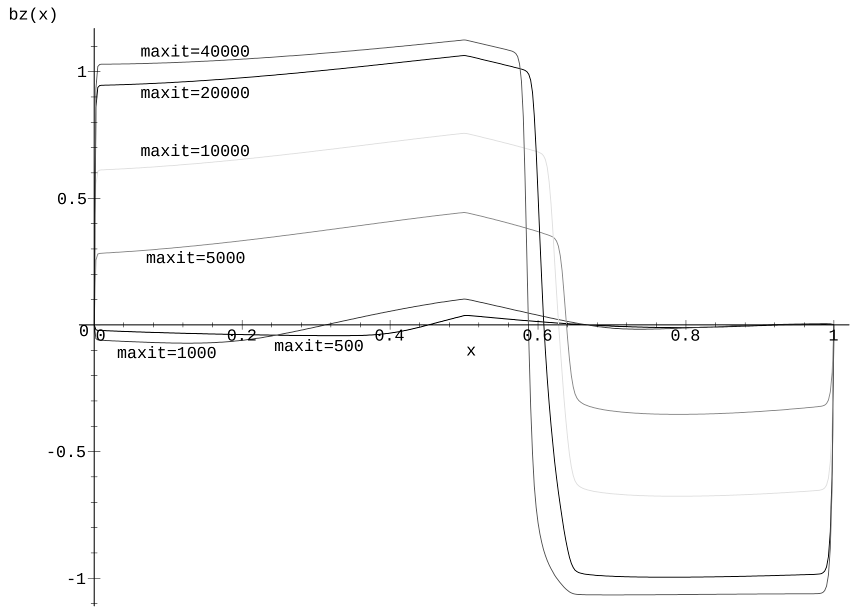

Fig. 6. Development of the gradient $bz(x)$.

Hence, 100% represents the memory usage of the basic reverse scheme, and 0.1% corresponds to $s = 10$ checkpoints. As one can see the checkpoint percentage can be pushed far below 10% before any impact on the run-time becomes apparent. Only when the percentage becomes lower than 1% we

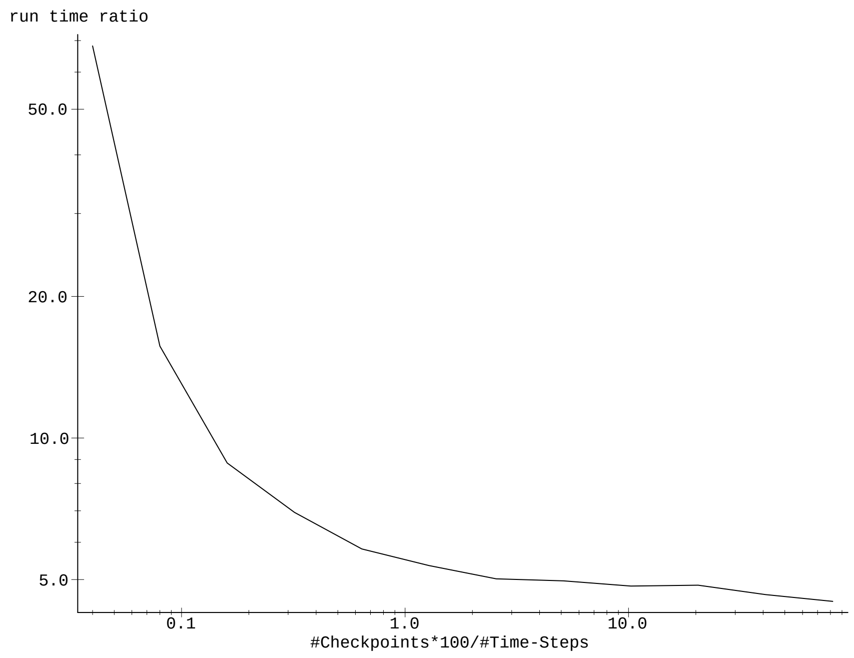

Fig. 7. Run-time for different numbers of checkpoints.

observe a significant increase in run-time. These threshold percentages are not constant and will decline as the total number of steps grows.

6. CONCLUSIONS

The checkpointing algorithm presented here allows various trade-offs between time and storage requirements. Specifically, logarithmic growth of the spatial complexity can be attained by selecting suitable parameters for `revolve`. In any case, a drastic reduction of the storage requirement can be achieved at a slight increase in the operations count. Hence, many applications that would require excessive storage without checkpointing can be solved using the function `revolve`. For example it has been successfully employed in seismic exploration [Symes 1997].

ACKNOWLEDGEMENTS

The authors are indebted to Andrew Mauer-Oats, Bijan Mohammadi, Dmitri Shiriaev, and Wolfgang Walter.

REFERENCES

GRIMM, J., POTTIER, L., AND ROSTAING-SCHMIDT, N. 1996. Optimal time and minimum space-time product for reversing a certain class of programs. In *Computational Differentiation: Techniques, Applications, and Tools*, M. Berz, C. Bischof, G. Corliss, and A. Griewank, Eds. SIAM, Philadelphia, PA, 161–172.

GRIEWANK, A. 1992. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optim. Meth. Softw. 1*, 1, 35–54.

GRIEWANK, A. 2000. *Evaluating Derivatives, Principles, and Techniques of Algorithmic Differentiation*. SIAM Frontiers in Applied Mathematics Series, vol. 19. SIAM, Philadelphia, PA.

RESTREPO, J. M., LEAF, G. K., AND GRIEWANK, A. 1998. Circumventing storage limitations in variational data assimilation studies. *SIAM J. Sci. Comput. 19*, 5, 1586–1605.

LEVEQUE, R. 1992. *Numerical Methods for Conservation*. Birkhäuser-Verlag, Basel, Switzerland.

LIONS, J. L. 1971. *Optimal Control of Systems Governed by Partial Differential Equations*. Springer-Verlag, New York, NY.

OSHER, S. AND SOLOMON, F. 1982. Upwind difference schemes for the hyperbolic systems of conservation laws. *Math. Comput. 38*, 158, 339–374.

SYMES, W. 1997. Framework for automatic differentiation applied to time-dependent problems. In *Proceedings of SIAM's 45th Anniversary Meeting* (Stanford, CA, July 14–18), SIAM, Philadelphia, PA.

WALTHER, A. 1999. Program reversal schedules for single- and multi-processor machines. Ph.D. Dissertation. Technical University Dresden, Dresden.

Received: January 1998; revised: October 1999; accepted: October 1999