Algorithmic Differentiation of Implicit Functions and Optimal Values

Bradley M. Bell[1] and James V. Burke[2]

[1] Applied Physics Laboratory, University of Washington, Seattle, WA 98195, USA, bradbell@washington.edu
[2] Department of Mathematics, University of Washington, Seattle, WA 98195, USA, burke@math.washington.edu

Summary. In applied optimization, an understanding of the sensitivity of the optimal value to changes in structural parameters is often essential. Applications include parametric optimization, saddle point problems, Benders decompositions, and multilevel optimization. In this paper we adapt a known automatic differentiation (AD) technique for obtaining derivatives of implicitly defined functions for application to optimal value functions. The formulation we develop is well suited to the evaluation of first and second derivatives of optimal values. The result is a method that yields large savings in time and memory. The savings are demonstrated by a Benders decomposition example using both the ADOL-C and CppAD packages. Some of the source code for these comparisons is included to aid testing with other hardware and compilers, other AD packages, as well as future versions of ADOL-C and CppAD. The source code also serves as an aid in the implementation of the method for actual applications. In addition, it demonstrates how multiple C++ operator overloading AD packages can be used with the same source code. This provides motivation for the coding numerical routines where the floating point type is a C++ template parameter.

Keywords: Automatic differentiation, Newton's method, iterative process, implicit function, parametric programming, C++ template functions, ADOL-C, CppAD

1 Introduction

In applications such as parametric programming, hierarchical optimization, Bender's decomposition, and saddle point problems, one is confronted with the need to understand the variational properties of an optimal value function. For example, in saddle point problems, one maximizes with respect to some variables and minimizes with respect to other variables. One may view a saddle point problem as a maximization problem where the objective is an optimal value function. Both first and second derivatives of the optimal value function are useful in solving this maximization problem. A similar situation occurs in the context of a Bender's decomposition (as was the case that motivated this research). In most cases, the optimal value is

evaluated using an iterative optimization procedure. Direct application of Algorithmic Differentiation (AD) to such an evaluation differentiates the entire iterative process (the direct method). The convergence theory of the corresponding derivatives is discussed in [2, 6, 7]. We review an alternative strategy that applies the implicit function theorem to the first-order optimality conditions. This strategy also applies, more generally, to differentiation of functions defined implicitly by a system of nonlinear equations. These functions are also evaluated by iterative procedures and the proposed method avoids the need to differentiate the entire iterative process in this context as well. The use of the implicit function theorem in this context is well known in the AD literature, e.g., [1, 4, 5, 10]. We provide a somewhat different formulation that introduces an auxiliary variable which facilitates the computation of first and second derivatives for optimal value functions.

In Sect. 2, the implicit function theorem is used to show that differentiating one Newton iteration is sufficient thereby avoiding the need to differentiate the entire iterative process. As mentioned above, this fact is well known in the AD literature. Methods for handling the case where the linear equations corresponding to one Newton step cannot be solved directly and require an iterative process are considered in [8]. An important observation is that, although the Newton step requires the solution of linear equations, the inversion of these equations need not be differentiated.

Consider the parametrized family of optimization problems

$$\mathscr{P}(x) \quad \text{minimize } F(x,y) \text{ with respect to } y \in \mathbb{R}^m$$

where $F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is twice continuously differentiable. Suppose that there is an open set $\mathscr{U} \subset \mathbb{R}^n$ such that for each value of $x \in \mathscr{U}$ it is possible to compute the optimal value for $\mathscr{P}(x)$ which we denote by $V(x)$. The function $V : \mathscr{U} \to \mathbb{R}$ defined in this way is called the optimal value function for the family of optimization problems $\mathscr{P}(x)$. In Sect. 3 we present a method for computing the derivative and Hessian of $V(x)$. This method facilitates using reverse mode to obtain the derivative of $V(x)$ in a small multiple of the work to compute $F(x,y)$. In Sect. 4 we present a comparison between differentiation of the entire iterative process (the direct method) with our suggested method for computing the Hessian. This comparison is made using the ADOL-C [9, version 1.10.2] and CppAD [3, version 20071225] packages.

The Appendix contains some of the source code that is used for the comparisons. This source code can be used to check the results for various computer systems, other AD packages, as well as future versions of ADOL-C and CppAD. It also serves as a starting point to implement the method for actual applications, i.e., other definitions of $F(x,y)$. In addition, it demonstrates the extent to which multiple C++ AD operator overloading packages can be used with the same C++ source code. This provides motivation for the coding numerical routines where the floating point type is a template parameter.

2 Jacobians of an Implicit Function

We begin by building the necessary tools for the application of AD to the differentiation of implicitly defined functions. Suppose $\mathscr{U} \subset \mathbb{R}^n$ and $\mathscr{V} \subset R^m$ are open and the function $H : \mathscr{U} \times \mathscr{V} \to \mathbb{R}^m$ is smooth. We assume that for each $x \in \mathscr{U}$ the equation $H(x,y) = 0$ has a unique solution $Y(x) \in \mathscr{V}$. That is, the equation $H(x,y) = 0$ implicitly defines the function $Y : \mathscr{U} \to \mathscr{V}$ by

$$H[x, Y(x)] = 0 \,.$$

Let $Y^{(k)}(x)$ denote the k-th derivative of Y and let $H_y(x,y)$ denote the partial derivative of H with respect to y. Conditions guaranteeing the existence of the function Y, as well as its derivatives and their formulas, in terms of the partials of H are given by the implicit function theorem. We use these formulas to define a function $\tilde{Y}(x,u)$ whose partial derivative in u evaluated at x gives $Y^{(1)}(x)$. The form of the function $\tilde{Y}(x,u)$ is based on the Newton step used to evaluate $Y(x)$. The partial derivative of $\tilde{Y}(x,u)$ with respect to u is well suited to the application of AD since it avoids the need to differentiate the iterative procedure used to compute $Y(x)$. The following theorem is similar to, e.g., [10, equation (3.6)] and [5, Lemma 2.3]. We present the result in our notation as an aid in understanding this paper.

Theorem 1. *Suppose $\mathscr{U} \subset \mathbb{R}^n$ and $\mathscr{V} \subset R^m$ are open, $H : \mathscr{U} \times \mathscr{V} \to \mathbb{R}^m$ is continuously differentiable, $\bar{x} \in \mathscr{U}$, $\bar{y} \in \mathscr{V}$, $H[\bar{x},\bar{y}] = 0$, and $H_y[\bar{x},\bar{y}]$ is invertible. Then, if necessary, $\mathscr{U}$ and $\mathscr{V}$ may be chosen to be smaller neighborhoods of $\bar{x}$ and $\bar{y}$, respectively, in order to guarantee the existence of a continuously differentiable function $Y : \mathscr{U} \to \mathscr{V}$ satisfying $Y(\bar{x}) = \bar{y}$ and for all $x \in \mathscr{U}$, $H[x, Y(x)] = 0$. Moreover, the function $\tilde{Y} : \mathscr{U} \times \mathscr{U} \to \mathbb{R}^m$, defined by*

$$\tilde{Y}(x,u) = Y(x) - H_y[x, Y(x)]^{-1} H[u, Y(x)] \,,$$

satisfies $\tilde{Y}(x,x) = Y(x)$ and

$$\tilde{Y}_u(x,x) = Y^{(1)}(x) = -H_y[x, Y(x)]^{-1} H_x[x, Y(x)] \,.$$

Note that $Y^{(1)}(x)$ can be obtained without having to completely differentiate the procedure for solving the linear equation $H_y[x, Y(x)]\Delta y = H[u, Y(x)]$. Solving this equation typically requires the computation of an appropriate factorization of the matrix $H_y[x, Y(x)]$. By using the function $\tilde{Y}$ one avoids the need to apply AD to the computation of this factorization. (This has been noted before, e.g., [10, equation (3.6)] and [5, Algorithm 3.1].)

As stated above, the function $\tilde{Y}$ is connected to Newton's method for solving the equation $H[x,y] = 0$ for y given x. For a given value for x, Newton's method (in its simplest form) approximates the value of $Y(x)$ by starting with an initial value $y_0(x)$ and then computing the iterates

$$y_{k+1}(x) = y_k(x) - H_y[x, y_k(x)]^{-1} H[x, y_k(x)]$$

until the value $H[x, y_k(x)]$ is sufficiently close to zero. The initial iterate $y_0(x)$ need not depend on x. The last iterate $y_k(x)$ is the value used to approximate $Y(x)$. If one directly applies AD to differentiate the relation between the final $y_k(x)$ and x (the direct method), all of the computations for all of the iterates are differentiated. Theorem 1 shows that one can alternatively use AD to compute the partial derivative $\tilde{Y}_u(x,u)$ at $u = x$ to obtain $Y^{(1)}(x)$. Since x is a fixed parameter in this calculation, no derivatives of the matrix inverse of $H_y[x, Y(x)]$ are required (in actual computations, this matrix is factored instead of inverted and the factorization need not be differentiated).

3 Differentiating an Optimal Value Function

Suppose that $\mathscr{U} \subset \mathbb{R}^n$ and $\mathscr{V} \subset \mathbb{R}^m$ are open, $F : \mathscr{U} \times \mathscr{V} \to \mathbb{R}$ is twice continuously differentiable on $\mathscr{U} \times \mathscr{V}$, and define the optimal value function $V : \mathscr{U} \to \mathbb{R}$ by

$$V(x) = \min \ F(x,y) \ \text{ with respect to } y \in \mathscr{V} \ . \tag{1}$$

In the next result we define a function $\tilde{V}(x,u)$ that facilitates the application of AD to the computation of the first and second derivatives of $V(x)$.

Theorem 2. *Let $\mathscr{U}$, $\mathscr{V}$, F and V be as in* (1)*, and suppose that $\bar{x} \in \mathscr{U}$ and $\bar{y} \in \mathscr{V}$ are such that $F_y(\bar{x},\bar{y}) = 0$ and $F_{yy}(\bar{x},\bar{y})$ is positive definite. Then, if necessary, $\mathscr{U}$ and $\mathscr{V}$ may be chosen to be smaller neighborhoods of $\bar{x}$ and $\bar{y}$, respectively, so that there exists a twice continuously differentiable function $Y : \mathscr{U} \to \mathscr{V}$ where $Y(x)$ is the unique minimizer of $F(x,\cdot)$ on $\mathscr{V}$, i.e.,*

$$Y(x) = \operatorname{argmin} F(x,y) \text{ with respect to } y \in \mathscr{V}.$$

We define $\tilde{Y} : \mathscr{U} \times \mathscr{U} \to \mathbb{R}$ and $\tilde{V} : \mathscr{U} \times \mathscr{U} \to \mathbb{R}$ by

$$\begin{aligned} \tilde{Y}(x,u) &= Y(x) - F_{yy}[x, Y(x)]^{-1} F_y[u, Y(x)] \ , \\ \tilde{V}(x,u) &= F[u, \tilde{Y}(x,u)] \ . \end{aligned}$$

It follows that for all $x \in \mathscr{U}$, $\tilde{V}(x,x) = V(x)$,

$$\begin{aligned} \tilde{V}_u(x,x) &= V^{(1)}(x) = F_x[x, Y(x)] \ , \\ \tilde{V}_{uu}(x,x) &= V^{(2)}(x) = F_{xx}[x, Y(x)] + F_{xy}[x, Y(x)] Y^{(1)}(x) \ . \end{aligned}$$

Proof. The implicit function theorem guarantees the existence and uniqueness of the function Y satisfying the first- and second-order sufficiency conditions for optimality in the definition of $V(x)$. It follows from the first-order necessary conditions for optimality that $F_y[x, Y(x)] = 0$. Defining $H(x,y) = F_y(x,y)$ and applying Theorem 1, we conclude that

$$\begin{aligned} \tilde{Y}(x,x) &= Y(x) \ , \\ \tilde{Y}_u(x,u) &= Y^{(1)}(x) \ . \end{aligned} \tag{2}$$

It follows that

$$\tilde{V}(x,x) = F[x,\tilde{Y}(x,x)] = F[x,Y(x)] = V(x) ,$$

which establishes the function value assertion in the theorem.

The definition of $\tilde{V}$ gives

$$\tilde{V}_u(x,u) = F_x[u,\tilde{Y}(x,u)] + F_y[u,\tilde{Y}(x,u)]\tilde{Y}_u(x,u) .$$

Using $F_y[x,Y(x)] = 0$ and $\tilde{Y}(x,x) = Y(x)$, we have

$$\tilde{V}_u(x,x) = F_x[x,Y(x)] . \tag{3}$$

On the other hand, since $V(x) = F[x,Y(x)]$, we have

$$\begin{aligned} V^{(1)}(x) &= F_x[x,Y(x)] + F_y[x,Y(x)]Y^{(1)}(x) , \\ &= F_x[x,Y(x)] . \end{aligned} \tag{4}$$

Equations (3) and (4) establish the first derivative assertions in the theorem.

The second derivative assertions requires a more extensive calculation. It follows from (4) that

$$V^{(2)}(x) = F_{xx}[x,Y(x)] + F_{xy}[x,Y(x)]Y^{(1)}(x) . \tag{5}$$

As noted above $F_y[x,Y(x)] = 0$ for all $x \in \mathscr{U}$. Taking the derivative of this identity with respect to x, we have

$$\begin{aligned} 0 &= F_{yx}[x,Y(x)] + F_{yy}[x,Y(x)]Y^{(1)}(x) , \\ 0 &= Y^{(1)}(x)^{\mathrm{T}} F_{yx}[x,Y(x)] + Y^{(1)}(x)^{\mathrm{T}} F_{yy}[x,Y(x)]Y^{(1)}(x) . \end{aligned} \tag{6}$$

Fix $x \in \mathscr{U}$ and define $G : \mathbb{R}^n \to \mathbb{R}^{n+m}$ by

$$G(u) = \begin{pmatrix} u \\ \tilde{Y}(x,u) \end{pmatrix} .$$

It follows from this definition that

$$\begin{aligned} \tilde{V}(x,u) &= (F \circ G)(u) , \\ \tilde{V}_u(x,u) &= F^{(1)}[G(u)]G^{(1)}(u) , \text{ and} \\ \tilde{V}_{uu}(x,u) &= G^{(1)}(u)^{\mathrm{T}} F^{(2)}[G(u)]G^{(1)}(u) \\ &\quad + \sum_{j=1}^{n} F_{x(j)}[G(u)]G^{(2)}_{(j)}(u) + \sum_{i=1}^{m} F_{y(i)}[G(u)]G^{(2)}_{(n+i)}(u) , \end{aligned}$$

where $F_{x(j)}$ and $F_{y(i)}$ are the partials of F with respect to x_j and y_i respectively, and where $G_{(j)}(u)$ and $G_{(n+i)}(u)$ are the j-th and $(n+i)$-th components of $G(u)$ respectively. Using the fact that $G^{(2)}_{(j)}(u) = 0$ for $j \le n$, $F_y[G(x)] = F_y[x,\tilde{Y}(x,x)] = 0$, $\tilde{Y}(x,x) = Y(x)$, and $\tilde{Y}_u(x,x) = Y^{(1)}(x)$, we have

$$\begin{aligned}\tilde{V}_{uu}(x,x) &= G^{(1)}(x)^{\mathrm{T}} F^{(2)}[G(x)] G^{(1)}(x) \,, \\ &= F_{xx}[x,Y(x)] + Y^{(1)}(x)^{\mathrm{T}} F_{yy}[x,Y(x)] Y^{(1)}(x) \\ &\quad + Y^{(1)}(x)^{\mathrm{T}} F_{yx}[x,Y(x)] + F_{xy}[x,Y(x)] Y^{(1)}(x) \,.\end{aligned}$$

We now use (6) to conclude that

$$\tilde{V}_{uu}(x,x) = F_{xx}[x,Y(x)] + F_{xy}[x,Y(x)] Y^{(1)}(x) \,.$$

This equation, combined with (5), yields the second derivative assertions in the theorem.

Remark 1. The same proof works when the theorem is modified with following replacements: $F_{yy}(\bar{x},\bar{y})$ is invertible, $Y : \mathscr{U} \to \mathscr{V}$ is defined by $F_y[x,Y(x)] = 0$, and $V : \mathscr{U} \to \mathbb{R}$ is defined by $V(x) = F[x,Y(x)]$. This extension is useful in certain applications, e.g., mathematical programs with equilibrium constraints (MPECs).

In summary, algorithmic differentiation is used to compute $\tilde{V}_u(x,u)$ and $\tilde{V}_{uu}(x,u)$ at $u = x$. The result is the first and second derivatives of the optimal value function $V(x)$, respectively. Computing the first derivative $V^{(1)}(x)$ in this way requires a small multiple of w where w is the amount of work necessary to compute values of the function $F(x,y)$ ([5, Algorithm 3.1] can also be applied to obtain this result). Computing the second derivative $V^{(2)}(x)$ requires a small multiple of nw (recall that n is the number of components in the vector x).

Note that one could use (2) to compute $Y^{(1)}(x)$ and then use the formula

$$V^{(2)}(x) = F_{xx}[x,Y(x)] + F_{xy}[x,Y(x)] Y^{(1)}(x)$$

to compute the second derivative of $V(x)$. Even if m is large, forward mode can be used to compute $\tilde{Y}_u(x,u)$ at $u = x$ in a small multiple of nw. Thus, it is possible that, for some problems, this alternative would compare reasonably well with the method proposed above.

4 Example

The direct method for computing $V^{(2)}(x)$ applies AD to compute the second derivative of $F[x,Y(x)]$ with respect x. This includes the iterations used to determine $Y(x)$ in the AD calculations. In this section, we present an example that compares the direct method to the method proposed in the previous section using both the ADOL-C [9, version 1.10.2] and CppAD [3, version 20071225] packages.

Our example is based on the function $\widehat{F} : U \times V \to \mathbb{R}$ defined by

$$\widehat{F}(x,y) = x \exp(y) + \exp(-y) - \log(x),$$

where $n = 1$ and $U \subset \mathbb{R}^n$ is the set of positive real numbers, $m = 1$ and $V \subset \mathbb{R}^m$ is the entire set of real numbers. This example has the advantage that the relevant

mathematical objects have closed form expressions. Indeed, $\widehat{Y}(x)$ (the minimizer of $\widehat{F}(x,y)$ with respect to y) and $\widehat{V}^{(2)}(x)$ (the Hessian of $\widehat{F}[x,\widehat{Y}(x)]$) are given by

$$\begin{aligned} \widehat{Y}(x) &= \log\left(1/\sqrt{x}\right) , \\ \widehat{V}(x) &= 2\sqrt{x} - \log(x) , \\ \widehat{V}^{(1)}(x) &= x^{-1/2} - x^{-1} , \text{ and} \\ \widehat{V}^{(2)}(x) &= x^{-2} - x^{-3/2}/2 . \end{aligned} \tag{7}$$

$$\widehat{V}^{(2)}(x) = x^{-2} - x^{-3/2}/2 . \tag{8}$$

Using this example function, we build a family of test functions that can be scaled for memory use and computational load. This is done by approximating the exponential function $\exp(y)$ with its Mth degree Taylor approximation at the origin, i.e.,

$$\mathrm{Exp}(y) = 1 + y + y^2/2! + \cdots + y^M/M! .$$

As M increases, the complexity of the computation increases. For all of values of M greater than or equal twenty, we consider the functions $\mathrm{Exp}(y)$ and $\exp(y)$ to be equal (to near numerical precision) for y in the interval $[0,2]$. Using $\mathrm{Exp}(y)$, we compute $F(x,y)$ and its partial derivatives with respect to y as follows:

$$\begin{aligned} F(x,y) &= x\mathrm{Exp}(y) + 1/\mathrm{Exp}(y) - \log(x) , \\ F_y(x,y) &= x\mathrm{Exp}(y) - 1/\mathrm{Exp}(y) , \text{ and} \\ F_{yy}(x,y) &= x\mathrm{Exp}(y) + 1/\mathrm{Exp}(y) . \end{aligned}$$

The method used to compute $Y(x)$ does not matter, we only need to minimize $F(x,y)$ with respect to y. For this example, it is sufficient to solve for a zero of $F_y(x,y)$ (because F is convex with respect to y). Thus, to keep the source code simple, we use ten iterations of Newton's method to approximate $Y(x)$ as follows: for $k = 0,\ldots,9$

$$\begin{aligned} y_0(x) &= 1 , \\ y_{k+1}(x) &= y_k(x) - F_y[x, y_k(x)]/F_{yy}[x, y_k(x)] , \text{ and} \\ Y(x) &= y_{10}(x) . \end{aligned} \tag{9}$$

Note that this is only an approximate value for $Y(x)$, but we use it as if it were exact, i.e., as if $F_y[x,Y(x)] = 0$. We then use this approximation for $Y(x)$ to compute

$$\begin{aligned} V(x) &= F[x,Y(x)] , \\ \tilde{Y}(x,u) &= Y(x) - F_y[u,Y(x)]/F_{yy}[x,Y(x)] , \text{ and} \\ \tilde{V}(x,u) &= F[u,\tilde{Y}(x,u)] . \end{aligned}$$

The source code for computing the functions F, F_y, F_{yy}, $V(x)$, $\tilde{Y}$ and $\tilde{V}$ are included in Sect. 6.1. The Hessians $V^{(2)}(x)$ (direct method) and $\tilde{V}_{uu}(x,u)$ (proposed method) are computed using the ADOL-C function `hessian` and the CppAD `ADFun<double>` member function `Hessian`.

As a check that the calculations of $V^{(2)}(x)$ are correct, we compute $\widehat{Y}(x)$ and $\widehat{V}^{(2)}(x)$ defined in equations (7) and (8). The source code for computing the functions $\widehat{Y}(x)$ and $\widehat{V}^{(2)}(x)$ are included in Sect. 6.2. The example results for this correctness check, and for the memory and speed tests, are included in the tables below.

4.1 Memory and Speed Tables

In the memory and speed tables below, the first column contains the value of `M` corresponding to each row (the output value `M` is the highest order term in the power series approximation for $\exp(y)$). The next two columns, `n_xx` and `n_uu`, contain a measure of how much memory is required to store the results of a forward mode AD operation (in preparation for a reverse mode operation) for the corresponding computation of $V^{(2)}(x)$ and $\tilde{V}_{uu}(x,u)$, respectively. The next column `n_uu/xx` contains the ratio of `n_uu` divided by `n_xx`. The smaller the `n_uu/xx` the more computation favors the use of $\tilde{V}_{uu}(x,u)$ for the second derivative. The next two columns, `t_xx` and `t_uu`, contain the run time, in milliseconds, used to compute $V^{(2)}(x)$ and $\tilde{V}_{uu}(x,u)$ respectively. Note that, for both ADOL-C and CppAD, the computational graph was re-taped for each computation of $V^{(2)}(x)$ and each computation of $\tilde{V}_{uu}(x,u)$. The next column `t_uu/xx` contains the ratio of `t_uu` divided by `t_xx`. Again, the smaller the ratio the more the computation favors the use of $\tilde{V}_{uu}(x,u)$ for the second derivative.

4.2 Correctness Tables

In the correctness tables below, the first column displays `M` corresponding to the correctness test, the second column displays $Y(x)$ defined by (9) ($x = 2$), the third column displays `Ycheck` which is equal to $\widehat{Y}(x)$ (see Sect. 6.2), the fourth column displays $V^{(2)}(x)$ computed by the corresponding AD package using the direct method, the fifth column displays $\tilde{V}_{uu}(x,u)$ computed by the corresponding AD package ($x = 2$, $u = 2$), the sixth column displays `V2check` which is equal to $\widehat{V}^{(2)}(x)$ (see Sect. 6.2).

4.3 System Description

The results below were generated using version 1.10.2 of ADOL-C, version 20071225 of CppAD, version 3.4.4 of the cygwin g++ compiler with the -O2 and -DNDEBUG compiler options, Microsoft Windows XP, a 3.00GHz pentium processor with 2GB of memory. The example results will vary depending on the operating system, machine, C++ compiler, compiler options, and hardware used.

4.4 ADOL-C

In this section we report the results for the case where the ADOL-C package is used. The ADOL-C usrparms.h values `BUFSIZE` and `TBUFSIZE` were left at their default value, 65536. The Hessians of $V(x)$ with respect to x and $\tilde{V}(x,u)$ with respect to u were computed using the ADOL-C function `hessian`. Following the call `hessian(tag,1,x,H)` a call was made to `tapestats(tag,counts)`. In the output below, `n_xx` and `n_uu` are the corresponding value `counts[3]`. This is an indication of the amount of memory required for the Hessian calculation (see [9] for more details).

```
Memory and Speed Table / ADOL-C
   M     n_xx     n_uu  n_uu/xx     t_xx     t_uu  t_uu/xx
  20     3803      746    0.196    0.794    0.271    0.341
  40     7163     1066    0.149   11.236    0.366    0.033
  60    10523     1386    0.132   15.152    0.458    0.030
  80    13883     1706    0.123   20.408    0.557    0.027
 100    17243     2026    0.117   25.000    0.656    0.026
Correctness Table / ADOL-C
   M       Y(x)     Ycheck       V_xx       V_uu    V2check
 100 -0.3465736 -0.3465736  0.0732233  0.0732233  0.0732233
```

4.5 CppAD

In this section we report the results for the case where the CppAD package is used. The Hessians of $V(x)$ with respect to x and $\tilde{V}(x,u)$ with respect to u were computed using the CppAD `ADFun<double>` member function `Hessian`. In the output below, `n_xx` and `n_uu` are the corresponding number of variables used during the calculation, i.e., the return value of `f.size_var()` where `f` is the corresponding AD function object. This is an indication of the amount of memory required for the Hessian calculation (see [3] for more details).

```
Memory and Speed Table / CppAD
   M     n_xx     n_uu  n_uu/xx     t_xx     t_uu  t_uu/xx
  20     2175      121    0.056    0.549    0.141    0.257
  40     4455      241    0.054    0.946    0.208    0.220
  60     6735      361    0.054    1.328    0.279    0.210
  80     9015      481    0.053    1.739    0.332    0.191
 100    11295      601    0.053    2.198    0.404    0.184
Correctness Table / CppAD
   M       Y(x)     Ycheck       V_xx       V_uu    V2check
 100 -0.3465736 -0.3465736  0.0732233  0.0732233  0.0732233
```

5 Conclusion

Theorem 1 provides a representation of an implicit function that facilitates efficient computation of its first derivative using AD. Theorem 2 provides a representation of an optimal value functions that facilitates efficient computation of its first and second derivative using AD. Section 4 demonstrates the advantage of this representation when using ADOL-C and CppAD. We suspect much smaller run times for CppAD, as compared to ADOL-C, are due to the fact that ADOL-C uses disk to store its values when the example parameter M is larger than 20. The source code for the example, that is not specific to a particular AD package, has been included as an aid in testing with other hardware and compilers, other AD packages, as well as future versions of ADOL-C, CppAD. It also serves as an example of the benefit of C++ template functions in the context of AD by operator overloading.

6 Appendix

6.1 Template Functions

The following template functions are used to compute $V^{(2)}(x)$ and $\tilde{V}_{uu}(x,u)$ using the ADOL-C type `adouble` and the CppAD type `CppAD::AD<double>`.

```
// Exp(x), a slow version of exp(x)
extern size_t M_;
template<class Float> Float Exp(const Float &x)
{   Float sum = 1., term = 1.;
    for(size_t i = 1 ; i < M_; i++)
    {   term *= ( x / Float(i) );
        sum  += term;
    }
    return sum;
}
// F(x, y) = x * exp(y) + exp(-y) - log(x)
template<class Float> Float F(const Float &x, const Float &y)
{   return x * Exp(y) + 1./Exp(y) - log(x); }
// F_y(x, y) = x * exp(y) - exp(-y)
template<class Float> Float F_y(const Float &x, const Float &y)
{   return x * Exp(y) - 1./Exp(y); }
// F_yy(x, y) = x * exp(y) + exp(-y)
template<class Float> Float F_yy(const Float &x, const Float &y)
{   return x * Exp(y) + 1./Exp(y); }
// Use ten iterations of Newtons method to compute Y(x)
template<class Float> Float Y(const Float &x)
{   Float y = 1.;                       // initial y
    for(size_t i = 0; i < 10; i++)      // 10 Newton iterations
        y = y - F_y(x, y) / F_yy(x, y);
    return y;
}
// V(x)
template<class Float> Float V(const Float &x)
{   return F(x, Y(x)); }
// Y~ (x , u), pass Y(x) so it does not need to be recalculated
template<class Float>
Float Ytilde(double x, const Float &u_ad, double y_of_x)
{   Float y_of_x_ad = y_of_x;
    return y_of_x_ad - F_y(u_ad , y_of_x_ad) / F_yy(x, y_of_x);
}
// V~ (x, u), pass Y(x) so it does not need to be recalculated
template<class Float>
Float Vtilde(double x, const Float &u_ad, double y_of_x)
{   return F(u_ad , Ytilde(x, u_ad, y_of_x) ); }
```

6.2 Check Functions

The functions $\widehat{Y}(x)$ defined in (7) and $\widehat{V}^{(2)}(x)$ defined in (8) are coded below as `Ycheck` and `V2check` respectively. These functions are used to check that the value of $Y(x)$ and $V^{(2)}(x)$ are computed correctly.

```
double Ycheck(double x)
{   return - log(x) / 2.; }
double V2check(double x)
{   return 1. / (x * x) - 0.5 / (x * sqrt(x)); }
```

Acknowledgement. This research was supported in part by NIH grant P41 EB-001975 and NSF grant DMS-0505712.

References

1. Azmy, Y.: Post-convergence automatic differentiation of iterative schemes. Nuclear Science and Engineering **125**(1), 12–18 (1997)
2. Beck, T.: Automatic differentiation of iterative processes. Journal of Computational and Applied Mathematics **50**(1–3), 109–118 (1994)
3. Bell, B.: CppAD: a package for C++ algorithmic differentiation (2007). `http://www.coin-or.org/CppAD`
4. Büskens, C., Griesse, R.: Parametric sensitivity analysis of perturbed PDE optimal control problems with state and control constraints. Journal of Optimization Theory and Applications **131**(1), 17–35 (2006)
5. Christianson, B.: Reverse accumulation and implicit functions. Optimization Methods and Software **9**, 307–322 (1998)
6. Gilbert, J.: Automatic differentiation and iterative processes. Optimization Methods and Software **1**(1), 13–21 (1992)
7. Griewank, A., Bischof, C., Corliss, G., Carle, A., Williamson, K.: Derivative convergence for iterative equation solvers. Optimization Methods and Software **2**(3-4), 321–355 (1993)
8. Griewank, A., Faure, C.: Reduced functions, gradients and Hessians from fixed-point iterations for state equations. Numerical Algorithms **30**(2), 113–39 (2002)
9. Griewank, A., Juedes, D., Mitev, H., Utke, J., Vogel, O., Walther, A.: ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. Tech. rep., Institute of Scientific Computing, Technical University Dresden (1999). Updated version of the paper published in *ACM Trans. Math. Software* 22, 1996, 131–167
10. Schachtner, R., Schaffler, S.: Critical stationary points and descent from saddlepoints in constrained optimization via implicit automatic differentiation. Optimization **27**(3), 245–52 (1993)