Optimization Methods and Software

On the Halley class of methods for unconstrained optimization problems
Haibin Zhang
College of Applied Science, Beijing University of Technology, Beijing, People's Republic of China

First published on: 13 July 2009

To cite this Article Zhang, Haibin(2010) 'On the Halley class of methods for unconstrained optimization problems', Optimization Methods and Software, 25: 5, 753 — 762, First published on: 13 July 2009 (iFirst)
To link to this Article DOI: 10.1080/10556780902951643
URL: http://dx.doi.org/10.1080/10556780902951643

PLEASE SCROLL DOWN FOR ARTICLE
On the Halley class of methods for unconstrained optimization problems

Haibin Zhang*

College of Applied Science, Beijing University of Technology, 100124 Beijing, People’s Republic of China

(Received 6 November 2008; final version received 6 April 2009)

Third-order methods can be used to solve efficiently the unconstrained optimization problems, and they, in most cases, use fewer iterations but more computational cost per iteration than a second-order method to reach the same accuracy. Recently, it has been shown by an article that under some conditions the ratio of the number of arithmetic operations of a third-order method (the Halley class of methods) and Newton’s method is constant (at most 5) per iteration. Automatic differentiation (AD) can compute fast and accurate derivatives such as the Jacobian, Hessian matrix and the tensor of the function. The Halley class of methods includes these high-order derivatives. In this paper, we apply AD efficiently to the methods and investigate the computational complexity of them. The results show that under general conditions even including the computation of the function and its derivative terms, the upper bound of the ratio can be reduced to 3.5.

Keywords: automatic differentiation; Halley method; Newton method; unconstrained optimization

AMS Subject Classification: 90C30; 65K05

1. Introduction

Consider the unconstrained optimization problem

$$\min_{x \in \mathbb{R}^n} f(x),$$

which satisfies the following assumptions.

**Assumption 1** $f$ is four-times continuously differentiable in a neighbourhood of the local minimum $x^*$ which minimizes $f(x)$.

**Assumption 2** The Hessian $\nabla^2 f(x^*)$ is positive definite.

We solve problem (1) with the third-order methods [6,7,9,10,12,16,17].

Since the products of the tensor and vectors are involved in the paper, the following notations are introduced and defined:

$$\nabla^3 f(x) u \triangleq \left( \sum_{k=1}^{n} \frac{\partial^3 f(x)}{\partial x_i \partial x_j \partial x_k} u_k \right) \in \mathbb{R}^{n \times n}.$$

*Email: zhanghaibin@bjut.edu.cn

ISSN 1055-6788 print / ISSN 1029-4937 online
© 2010 Taylor & Francis
DOI: 10.1080/10556780902951643
http://www.informaworld.com
\[ \nabla^3 f(x)uv \overset{\text{def}}{=} (\nabla^3 f(x)u)v \in \mathbb{R}^n, \]

where \( u, v \in \mathbb{R}^n \) and \( i, j, k = 1, \ldots, n \).

The Halley class of methods can be written as

\[ x_{k+1} = x_k + s_1^k + s_2^k, \]

where \( s_1^k \) and \( s_2^k \) are, respectively, the solutions to the Newton equation

\[ \nabla^2 f(x_k)s_1^k = -\nabla f(x_k), \]

and the Halley equation

\[ (\nabla^2 f(x_k) + \alpha \nabla^3 f(x_k)s_1^k)s_2^k = -\frac{1}{2} \nabla^3 f(x_k)s_1^k s_1^k. \]

For \( \alpha = 0 \) the method is Chebyshev’s method (the improved method of tangent hyperbolas, [9,18]). When \( \alpha = 1/2 \) the method is the original Halley’s method, see [16] and [17]. For \( \alpha = 1 \) the method is referred to as the super Halley method, see [8]. Recent research on the Halley method include [4,6,7], etc.

The Halley class of methods and the Newton method are famous high-order optimization methods, in general, they have third-order and second-order convergence, respectively. In theory, the former can use fewer iterations than the latter to reach the same accuracy. However, they may use more computational complexity per iteration than the latter. That is chiefly because the Halley class of methods needs to compute the tensor term of the target function in Equation (4).

By traditional differentiation methods (symbolic differentiation, divided difference, etc.), it has a very large computational cost to evaluate the tensor terms when \( n \gg 1 \), and it is about \( O(n) \) times that to evaluate the Hessian matrix. Therefore, the ratio of the computational complexities per iteration of Halley over Newton increases unlimitedly with the problem dimension \( n \to +\infty \).

Recently, Gundersen and Steihaug [6] showed that under some conditions the ratio of the number of arithmetic operations of a third-order method (the Halley class of methods) and Newton’s method is constant per iteration, that is

\[ \frac{5 - 9}{7} \leq \frac{\text{One Halley step}}{\text{One Newton step}} \leq 5. \]

The derivative matrices and tensors can be approximated or calculated by a variety of techniques including symbolic differentiation, divided difference, and automatic differentiation (AD), etc. AD is a set of techniques for transforming a program that calculates numerical values of a function into a program that calculates numerical values for derivatives of that function. The basic forward mode and reverse mode of AD were apparently first proposed in [13,15], respectively. Later, with the development of computer hardware and software technology, AD has developed rapidly and been applied to many fields [1,2,3,5,14]. AD can compute fast and accurate derivatives and is superior to traditional differentiation methods.

In this paper, we establish the Halley class of methods and Newton’s method with AD, investigate and compare the computational complexity of them. The results by theoretical analysis show that under general conditions, even including the computation of the function and its derivative terms, the upper bound of the ratio can be reduced to 3.5.

The remainder of the paper is arranged as follows: in Section 2, AD algorithms are recalled and the Halley class of methods with AD are established, and the computational complexity of them is analysed and derived. In Section 3, we shall discuss the implementation of the methods with AD and give the experimental results. Some conclusions are drawn in the last section.
2. Automatic differentiation algorithms and Halley class of methods

In this section, we establish the automatic differentiation algorithms and Halley class of methods, and discuss their computational complexity.

2.1 Automatic differentiation algorithms

Automatic differentiation can compute fast and accurate derivatives such as the gradient, the Jacobian matrix, and the Hessian matrix of the function. For instance, the cost to evaluate gradient $\nabla f(x)$ (where $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$) through the reverse mode is constant times that of the underlying functions. Furthermore, through the combination of the forward and the reverse modes, the cost to evaluate the Hessian term $\nabla^2 f(x) \dot{x}$ is merely several times that of the underlying functions, as well as the cubic tensor term $\nabla^3 f(x) \dot{x} \ddot{x}$, where $\dot{x}, \ddot{x} \in \mathbb{R}^n$. The AD algorithms used in this paper are described briefly as follows [5, Chapters 3 and 4], where Algorithm AD1, AD2, and AD3 are used to evaluate $\nabla f(x)$, $\nabla^2 f(x) \dot{x}$, and $\nabla^3 f(x) \dot{x} \ddot{x}$.

**Algorithm 1** Algorithm AD1

*Step 0.* Set $x \in \mathbb{R}^n$.

*Step 1.* Evaluate $f(x)$ using the general evaluation procedure.

*Step 2.* Evaluate $\nabla f(x)$ using reverse propagation of gradients.

**Algorithm 2** Algorithm AD2(m)

*Step 0.* Set $x, \dot{x}_1, \ldots, \dot{x}_m \in \mathbb{R}^n$.

*Step 1.* Calculate $\nabla f(x)$ by Algorithm AD1.

*Step 2.* Evaluate $\nabla^2 f(x) \cdot \dot{x}_i, i = 1, \ldots, m$, using forward propagation of tangents.

**Algorithm 3** Algorithm AD3

*Step 0.* Set $x, \dot{x} \in \mathbb{R}^n$.

*Step 1.* Calculate $\nabla^2 f(x) \dot{x}$ by Algorithm AD2.

*Step 2.* Evaluate $\nabla^3 f(x) \dot{x} \ddot{x}$, using forward propagation of tangents.

Consider the computation cost $Q^{AD1}$ to evaluate a gradient $\nabla f$ by Algorithm AD1. By (3.24) in [5, Chapter 3, Section 3.4], it has the upper bound $4Q_f$, and often the bound is tight, so it is reasonable to assume that

$$Q^{AD1} = 4Q_f, \quad (5)$$

where $Q_f$ is the computational complexity to compute one $f(x)$ value.

The computation cost $Q^{AD2}(m)$ of Algorithm AD2 consists of two parts: the cost $Q^{AD1}$ to evaluate a gradient $\nabla f$ and the cost $Q^{HV}(m)$ to evaluate $m$ Hessian-vector products after $\nabla f(x)$ is evaluated. Using the above Equation (5) and (3.6) in [5, Chapter 3, Section 3.2], and adopting its tight upper bound, therefore,

$$Q^{AD2}(m) = (1 + 1.5m)Q^{AD1} = Q^{AD1} + Q^{HV}(m) = (4 + 6m)Q_f, \quad (6)$$

where

$$Q^{HV}(m) = 1.5mQ^{AD1} = 6mQ_f. \quad (7)$$
Let $Q^{AD3}$ be the computation cost involved in Algorithm AD3 after the gradient $\nabla f(x)$ has been evaluated. Using (3.6) in [5, Chapter 3, Section 3.2], similarly, we can arrive at

$$Q^{AD3} = (1 + 1.5)Q^{AD2}(1) - Q^{AD1} = 25Q_f - 4Q_f = 21Q_f. \quad (8)$$

2.2 The Halley class of methods with automatic differentiation

Let us give the Halley class of methods, which are used to solve the unconstrained optimization problems with automatic differentiation.

**Algorithm 4 Algorithm HAD (the Halley class of methods with AD)**

**Step 0.** Initial data. Set the initial points $x_0 \in \mathbb{R}^n$.

**Step 1.** Evaluate $\nabla f(x_k)$ by Algorithm AD1. If $\nabla f(x_k) = 0$, then terminate the iteration by taking $x^* = x_k$.

**Step 2.** Evaluate $\nabla^2 f(x_k)$ by using Algorithm AD2 with setting $\hat{x}_i = e_i$, $i = 1, \ldots, n$, where $e_i$ is the $i$th Cartesian basic vector in $\mathbb{R}^n$.

2.1 Get $\nabla^2 f(x_k) = L_kD_kL_k^T$ by Cholesky factorization;

2.2 Solve $s_k^1$ in Equation (3) with $\nabla^2 f(x_k) = L_kD_kL_k^T$;

**Step 3.** If $\alpha = 0$, compute $\nabla^3 f(x_k)s_k^1s_k^1$ by Algorithm AD3 with $x = x_k$ and $\hat{x} = s_k^1$, go to Step 5.

**Step 4.** Computing $g(x_k) = \alpha \nabla^2 f(x_k)s_k^1$ by Algorithm AD2 with $\hat{x} = \alpha s_k^1$, then $\alpha \nabla^3 f(x_k)s_k^1$ by using Step 2 of Algorithm AD2 to $g(x)$ with setting $\hat{x}_i = e_i$, $i = 1, \ldots, n$, where $e_i$ is the $i$th Cartesian basic vector in $\mathbb{R}^n$. Finally, computing $G(x_k) = \nabla^2 f(x_k) + \alpha \nabla^3 f(x_k)s_k^1$ by matrices additions.

4.1 Get $\bar{L}_k\bar{D}_k\bar{L}_k^T$ by executing Cholesky factorization on $G(x_k)$;

4.2 Get $1/2\nabla^3 f(x_k)s_k^1s_k^1$ by computing $(1/2\alpha)(G(x_k)s_k^1 + \nabla f(x_k))$;

**Step 5.** Solve $s_k^2$ in Equations (4) with $G(x_k) = \bar{L}_k\bar{D}_k\bar{L}_k^T$;

**Step 6.** Update solution estimate. Set $x_{k+1} = x_k + s_k^1 + s_k^2$. Set $k = k + 1$, and go to Step 1.

In order to derive the main results, let us first give the following lemma.

**Lemma 1** Assume that $A, B, C,$ and $D > 0$, then $(A + B)/(C + D) \leq \max\{A/C, B/D\}$.

**Proof** Obviously, if $A/C \leq B/D$, then $(A + B)/(C + D) \leq B/D$ else $(A + B)/(C + D) \leq A/C$. Therefore, the conclusion is proved. \hfill \Box

**Theorem 1** If the matrices $\nabla^2 f(x)$ and $G(x)$ in Algorithm HAD ($\alpha \neq 0$) are positive definite and the Newton equation and Halley equation are solved by a direct method then the ratio of the number of arithmetic operations of the Halley class of methods and Newton’s method is constant per iteration, and the ratio

$$2 < R = \frac{\text{flops(OneHalleystep)}}{\text{flops(OneNewtonstep)}} < 3.5. \quad (9)$$

In addition, when $\alpha = 0$, the Halley method (Chebyshev’s method) satisfies that when $n \gg 1$

$$1 < R = \frac{\text{flops(OneHalleystep)}}{\text{flops(OneNewtonstep)}} \approx 1. \quad (10)$$

**Proof** For convenience in expression, we introduce the following notations:

$Q_f$ denotes the number of arithmetic operations to compute the function $f(x)$; $Q_D$ denotes the number of arithmetic operations to solve the linear equations by a direct method.
Let us first analyse the computational complexity of one step of Algorithm HAD.

(a) On the Newton Equation (3)

Using Algorithms AD1 and AD2 to compute $\nabla f(x)$ and $\nabla^2 f(x)$, by Equation (6) with $m = n$, the computational complexity is $(6n^2 + 6n + 4)Q_f$.

Cholesky factorization $\nabla^2 f(x) = L_k D_k L_k^T$ needs $(1/3)n^3 + (1/2)n^2 - (5/6)n$ arithmetic operations (see [7]). Solving the linear equations on the coefficient matrix $L_k D_k L_k^T$ needs $2n^2 - n$ arithmetic operations. Thus

$$Q_D = \frac{1}{3}n^3 + \frac{5}{2}n^2 - \frac{11}{6}n.$$  \hspace{1cm} (11)

Therefore, the total number of arithmetic operations of the Newton Equation (3) is

$$Q_{Newton} = Q_D + (6n + 4)Q_f.$$  \hspace{1cm} (12)

(b) On the Halley Equation (4)

To compute $g(x)$ in Step 4 of Algorithm HAD, we can get $g(x)$ using Algorithm AD2 with $\dot{x} = \alpha s_k^1$. By Equations (5) and (7), it needs $n + (1.5 + 1)Q_f = n + 10Q_f$.

We can obtain $G(x)$ by using Step 2 of Algorithm AD2 to $g(x)$ with setting $\dot{x}_i = e_i, i = 1, \ldots, n$, where $e_i$ is the $i$th Cartesian basic vector in $\mathbb{R}^n$. By Equation (7) with $m = n$, it needs $1.5n(10Q_f) = 15nQ_f$.

Similar to (a), the Cholesky factorization and solving the linear equations on the coefficient matrix $\tilde{L}_k \tilde{D}_k \tilde{L}_k^T$ need $Q_D$.

Compute $1/2 \nabla^3 f(x_k)s_k^1s_k^1 + \nabla f(x_k)$. By $\nabla^2 f(x_k)s_k^1 + \nabla f(x_k) = 0$ and $G(x_k) = \nabla^2 f(x_k) + \alpha \nabla^3 f(x_k)s_k^1$, we have

$$G(x_k)s_k^1 + \nabla f(x_k) = \{\nabla^2 f(x_k) + \alpha \nabla^3 f(x_k)s_k^1\}s_k^1 + \nabla f(x_k)$$

$$= \alpha \nabla^3 f(x_k)s_k^1s_k^1 + \nabla^2 f(x_k)s_k^1 + \nabla f(x_k) = \alpha \nabla^3 f(x_k)s_k^1s_k^1.$$

That is

$$\frac{1}{2} \nabla^3 f(x_k)s_k^1s_k^1 = \frac{1}{2\alpha} (G(x_k)s_k^1 + \nabla f(x_k)).$$

Therefore, it needs $n^2 + n$ after $G(x_k)$ and $\nabla f(x_k)$ are ready.

Therefore, the total number of arithmetic operations of one Halley Equation (4) is

$$Q_{Halley} = [Q_D + n^2 + n] + (15n + 10)Q_f.$$  \hspace{1cm} (13)

(c) On the ratio of the computational complexity of one step of the Halley method and one step of Newton’s method.

It is easy to get the ratio

$$R = \frac{\text{flops (One Halley step)}}{\text{flops (One Newton step)}} = \frac{Q_{Halley} + Q_{Newton}}{Q_{Newton}} = 1 + \frac{Q_{Halley}}{Q_{Newton}}$$

$$= 1 + \frac{[Q_D + n^2 + n] + (15n + 10)Q_f}{Q_D + (6n + 4)Q_f}.$$  \hspace{1cm} (14)
By Lemma 1 and
\[ 1 < \frac{Q_D + n^2 + n}{Q_D} < \frac{(15n + 10)Q_f}{(6n + 4)Q_f} = 2.5, \]
when \( n \gg 1 \), we have
\[ 2 < R = 1 + \frac{[Q_D + n^2 + n] + (15n + 10)Q_f}{Q_D + (6n + 4)Q_f} \]
\[ \leq 1 + \max \left\{ \frac{Q_D + n^2 + n}{Q_D}, \frac{(15n + 10)Q_f}{(6n + 4)Q_f} \right\} < 1 + 2.5 = 3.5. \] (15)

When \( \alpha = 0 \), Algorithm HAD executes Step 3 instead of Step 4. Computing \( 1/2 \nabla^3 f(x_k)s_k s_k^T \) by Algorithm AD3 with \( x = x_k \) and \( s = s_k \), by Equation (8), it needs \( 21Q_f + n \). Thus, Algorithm HAD needs \( Q_D + (6n + 4)Q_f + n + 21Q_f \) after Equation (3) is solved. Therefore, the ratio satisfies
\[ R = \frac{Q_D + (6n + 4)Q_f + n + 21Q_f}{Q_D + (6n + 4)Q_f} = \frac{Q_D + n + (6n + 25)Q_f}{Q_D + (6n + 4)Q_f}, \]
and by Lemma 1,
\[ R = \frac{Q_D + n + (6n + 25)Q_f}{Q_D + (6n + 4)Q_f} \]
\[ \leq \max \left\{ \frac{Q_D + n}{Q_D}, \frac{(6n + 25)Q_f}{(6n + 4)Q_f} \right\} \]
\[ = \max \left\{ 1 + \frac{n}{Q_D}, 1 + \frac{21}{6n + 4} \right\}. \] (16)

It is not difficult to obtain Equation (10) by (16).

3. Numerical experiments

In this section, we test the Halley class of methods and the Newton method with AD by all of the numerical examples in [18]: the Extended Rosenbrock function, the Penalty function I, the Extended Powell singular function, and the Discrete boundary value function. These test problems are quoted from the unconstrained optimization problems in [11].

Consider the unconstrained optimization problem (1) by setting
\[ f(x) = \sum_{i=1}^{m} f_i^2(x), \] (17)
where \( f_1, \ldots, f_m \) are given in the following problems, and we adopt the general format in [11]:

Name of function

(a) Dimensions
(b) Function definition
(c) Starting point (designated \( x_0 \))

For the convenience of comparison, we choose the following three problems.
Problem 1  The Extended Rosenbrock function

(a) $n$ variable but even, $m = n$
(b) $f_{2i-1}(x) = 10(x_{2i} - x_{2i-1}^2)$
(c) $x_0 = (\xi_j)$, where $\xi_{2j-1} = -1.2, \xi_{2j} = 1$

Problem 2  Penalty function I

(a) $n$ variable, $m = n + 1$
(b) $f_i(x) = a^{1/2}(x_j - 1), 1 \leq i \leq n$
$c_i(x) = (\sum_{j=1}^n x_j^2) - 1/4,$
$\text{where } a = 10^{-5}$
(c) $x_0 = (\xi_j)$, where $\xi_j = j$

Problem 3  The Extended Powell singular function

(a) $n$ variable but a multiple of $4$, $m = n$
(b) $f_{4i-3}(x) = x_{4i-3} + 10x_{4i-2}$
$f_{4i-2}(x) = 5^{1/2}(x_{4i-1} - x_{4i})$
$f_{4i-1}(x) = (x_{4i-2} - 2x_{4i-1})^2$
$f_{4i}(x) = 10^{1/2}(x_{4i-3} - x_{4i})^2$
(c) $x_0 = (\xi_j)$, where $\xi_{4j-3} = 0.03, \xi_{4j-2} = -0.01, \xi_{4j-1} = 0.01, \xi_{4j} = 0.01$

Problem 4  Discrete boundary value function

(a) $n$ variable, $m = n$
(b) $f_i(x) = 2x_i - x_{i-1} - x_{i+1} + h^2(x_i + t_i + 1)^3/2$,
$\text{where } h = 1/(n+1), t_i = ih, x_0 = x_{n+1} = 0$
(c) $x_0 = (\xi_j)$, where $\xi_j = t_j(t_j - 1)$

They are executed in C++ routines with double precision.

In addition, the condition

$$\|\nabla f(x_k)\| \leq 10^{-6}$$  \hspace{1cm} (18)

is used for termination test.

In Tables 1–4, the dimension $n = 200, 400, \ldots, 1000$ of the problem is listed in the first column, and the main values we are interested in are:

(1) The iteration numbers:

$N_{\text{ITER}}$ – the iteration number of the Newton method. Its value is reported in the second column.

$H^{\alpha}_{\text{ITER}}$ – the iteration numbers of Chebyshev’s method (the Halley method $\alpha = 0$), reported in the third column.

$H_{\text{ITER}}$ – the iteration numbers of the super Halley method ($\alpha = 1$), reported in the sixth column.

$H^{1/2}_{\text{ITER}}$ – the iteration numbers of the original Halley method ($\alpha = 1/2$), reported in the eighth column.
The ratio of CPU times which Chebyshev’s method and the Newton method use, that is

\[ R_{\text{TOTAL}}^C = \frac{\text{CPU time by Chebyshev’s method}}{\text{CPU time by the Newton method}} \]

is listed in the forth column.

The ratio of CPU times of the average one step using the Halley method and Newton method, that is

\[ R_{\text{AVER}}^M = \frac{(\text{CPU time by the Halley method})/H_{\text{ITER}}^M}{(\text{CPU time by the Newton method})/N_{\text{ITER}}} \]

where the superscript \( M \) can be \( C, S, \) or \( O \):
- \( R_{\text{AVER}}^C \) – that of Chebyshev’s method, listed in the fifth column;
- \( R_{\text{AVER}}^S \) – that of the super Halley method, listed in the seventh column;
- \( R_{\text{AVER}}^O \) – that of the original Halley method, listed in the last column.

The above problems are successfully solved. The norms of the termination gradients are not listed in the tables. All of the optima of the problems successfully solved are the ones reported in [11], here they are left out.

In addition, since general optimization problems (1) are involved in this paper, the special structures above four problems are not considered in this paper, including numerical experiments. That is, the implementation of AD does not utilize the sparsity in \( \nabla^2 f(x_k) \) (like using the CPR technique, see [5, Chapter 7]).

Tables 1–4 show that:

1. The values of the seventh and ninth column satisfy \( R_{\text{AVER}}^S, R_{\text{AVER}}^O < 3.5 \), and they are approximately the lower bound 2. It supports the conclusion (9) of Theorem 1.
2. Except for Problem 2, the values of \( R_{\text{AVER}}^C \approx 1 \). Therefore, it supports the result (10) of Theorem 1 in total.
   To make further comparison, we test Problems 1–4 with \( n = 2000, \ldots, 5000 \) by Chebyshev’s method and Newton’s method. The results are reported in Table 5. Together with Tables 1–4, they show that
3. \( H_{\text{ITER}}^C < N_{\text{ITER}}^C \) except for Problem 1 and \( R_{\text{TOTAL}}^C < 1 \) except for Problems 1 and 2 with \( n \leq 1000 \). They show that Chebyshev’s method is more efficient than Newton’s method because it uses less CPU time and fewer iteration in most cases.
4. The run time of the Halley class methods with \( \alpha \neq 0 \) exceeds the one of the Newton method because the additional Cholesky factorization that is required is too much to achieve a good run time but the evaluation of the third-order derivative on the right-hand side of Equation (4) is OK in terms of run time.

### Table 1. Extended Rosenbrock function (Problem 1).

<table>
<thead>
<tr>
<th>( n )</th>
<th>( N_{\text{ITER}} )</th>
<th>( H_{\text{ITER}}^C )</th>
<th>( R_{\text{TOTAL}}^C )</th>
<th>( R_{\text{AVER}}^C )</th>
<th>( H_{\text{ITER}}^S )</th>
<th>( R_{\text{AVER}}^S )</th>
<th>( H_{\text{ITER}}^O )</th>
<th>( R_{\text{AVER}}^O )</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>6</td>
<td>6</td>
<td>1.0000</td>
<td>1.0000</td>
<td>7</td>
<td>1.6714</td>
<td>7</td>
<td>1.6286</td>
</tr>
<tr>
<td>400</td>
<td>6</td>
<td>7</td>
<td>1.1700</td>
<td>1.0029</td>
<td>7</td>
<td>2.0057</td>
<td>7</td>
<td>2.0057</td>
</tr>
<tr>
<td>600</td>
<td>6</td>
<td>7</td>
<td>1.1768</td>
<td>1.0087</td>
<td>7</td>
<td>2.0096</td>
<td>7</td>
<td>2.0017</td>
</tr>
<tr>
<td>800</td>
<td>6</td>
<td>7</td>
<td>1.1734</td>
<td>1.0058</td>
<td>7</td>
<td>2.0048</td>
<td>7</td>
<td>2.0026</td>
</tr>
<tr>
<td>1000</td>
<td>6</td>
<td>7</td>
<td>1.1709</td>
<td>1.0036</td>
<td>7</td>
<td>2.0027</td>
<td>7</td>
<td>2.0015</td>
</tr>
</tbody>
</table>
Table 2. Penalty function I (Problem 2).

<table>
<thead>
<tr>
<th>n</th>
<th>N_ITER</th>
<th>H^C_{ITER}</th>
<th>R^C_{TOTAL}</th>
<th>R^C_{AVER}</th>
<th>H^S_{ITER}</th>
<th>R^S_{AVER}</th>
<th>H^O_{ITER}</th>
<th>R^O_{AVER}</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>33</td>
<td>28</td>
<td>1.6705</td>
<td>1.9687</td>
<td>29</td>
<td>1.9397</td>
<td>28</td>
<td>1.9955</td>
</tr>
<tr>
<td>400</td>
<td>35</td>
<td>27</td>
<td>1.5783</td>
<td>2.0460</td>
<td>28</td>
<td>2.0438</td>
<td>27</td>
<td>2.0762</td>
</tr>
<tr>
<td>600</td>
<td>36</td>
<td>26</td>
<td>1.4635</td>
<td>2.0264</td>
<td>29</td>
<td>2.0270</td>
<td>26</td>
<td>2.0513</td>
</tr>
<tr>
<td>800</td>
<td>37</td>
<td>27</td>
<td>1.4746</td>
<td>2.0207</td>
<td>28</td>
<td>2.0218</td>
<td>27</td>
<td>2.0453</td>
</tr>
<tr>
<td>1000</td>
<td>39</td>
<td>27</td>
<td>1.4009</td>
<td>2.0235</td>
<td>29</td>
<td>2.0218</td>
<td>27</td>
<td>2.0419</td>
</tr>
</tbody>
</table>

Table 3. Extended Powell singular function (Problem 3).

<table>
<thead>
<tr>
<th>n</th>
<th>N_ITER</th>
<th>H^C_{ITER}</th>
<th>R^C_{TOTAL}</th>
<th>R^C_{AVER}</th>
<th>H^S_{ITER}</th>
<th>R^S_{AVER}</th>
<th>H^O_{ITER}</th>
<th>R^O_{AVER}</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>7</td>
<td>5</td>
<td>0.7647</td>
<td>1.0706</td>
<td>5</td>
<td>1.8941</td>
<td>5</td>
<td>1.8941</td>
</tr>
<tr>
<td>400</td>
<td>8</td>
<td>5</td>
<td>0.6909</td>
<td>0.9744</td>
<td>6</td>
<td>1.9850</td>
<td>6</td>
<td>2.0250</td>
</tr>
<tr>
<td>600</td>
<td>8</td>
<td>6</td>
<td>0.7453</td>
<td>0.9938</td>
<td>6</td>
<td>1.9813</td>
<td>6</td>
<td>1.9969</td>
</tr>
<tr>
<td>800</td>
<td>8</td>
<td>6</td>
<td>0.7553</td>
<td>1.0070</td>
<td>6</td>
<td>1.9980</td>
<td>6</td>
<td>2.0020</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>6</td>
<td>0.7673</td>
<td>1.0230</td>
<td>6</td>
<td>1.9959</td>
<td>6</td>
<td>1.9993</td>
</tr>
</tbody>
</table>

Table 4. Discrete boundary value function (Problem 4).

<table>
<thead>
<tr>
<th>n</th>
<th>N_ITER</th>
<th>H^C_{ITER}</th>
<th>R^C_{TOTAL}</th>
<th>R^C_{AVER}</th>
<th>H^S_{ITER}</th>
<th>R^S_{AVER}</th>
<th>H^O_{ITER}</th>
<th>R^O_{AVER}</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>17</td>
<td>9</td>
<td>0.6000</td>
<td>1.1333</td>
<td>14</td>
<td>1.8545</td>
<td>13</td>
<td>1.8545</td>
</tr>
<tr>
<td>400</td>
<td>17</td>
<td>9</td>
<td>0.5235</td>
<td>1.0467</td>
<td>16</td>
<td>2.0251</td>
<td>15</td>
<td>1.9860</td>
</tr>
<tr>
<td>600</td>
<td>18</td>
<td>9</td>
<td>0.4999</td>
<td>1.0059</td>
<td>16</td>
<td>2.0121</td>
<td>16</td>
<td>1.9994</td>
</tr>
<tr>
<td>800</td>
<td>18</td>
<td>9</td>
<td>0.4992</td>
<td>1.1432</td>
<td>17</td>
<td>2.0109</td>
<td>16</td>
<td>1.9984</td>
</tr>
<tr>
<td>1000</td>
<td>18</td>
<td>9</td>
<td>0.4998</td>
<td>0.9996</td>
<td>17</td>
<td>2.0028</td>
<td>16</td>
<td>2.0026</td>
</tr>
</tbody>
</table>

Table 5. Chebyshev’s method vs Newton’s method with \( n = 2000, \ldots, 5000 \).

<table>
<thead>
<tr>
<th>n</th>
<th>N_ITER</th>
<th>H^C_{ITER}</th>
<th>R^C_{TOTAL}</th>
<th>( n )</th>
<th>N_ITER</th>
<th>H^C_{ITER}</th>
<th>R^C_{TOTAL}</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>6</td>
<td>7</td>
<td>1.1402</td>
<td>3000</td>
<td>6</td>
<td>7</td>
<td>1.1652</td>
</tr>
<tr>
<td>3000</td>
<td>6</td>
<td>7</td>
<td>1.1955</td>
<td>4000</td>
<td>6</td>
<td>7</td>
<td>1.1529</td>
</tr>
<tr>
<td>5000</td>
<td>6</td>
<td>7</td>
<td>0.7163</td>
<td>2000</td>
<td>8</td>
<td>6</td>
<td>0.7291</td>
</tr>
<tr>
<td>2000</td>
<td>8</td>
<td>6</td>
<td>0.7445</td>
<td>3000</td>
<td>8</td>
<td>6</td>
<td>0.6466</td>
</tr>
</tbody>
</table>

4. Conclusion

The Halley class of methods with AD is established in this paper, where the gradient, Hessian, and tensor terms are computed by AD. We compare the Halley class of methods with Newton’s method in the computational complexity per iteration in theory and limited numerical experiments. In connection with [6,7], the results show that the Halley class of methods is competitive with the development of automatic differentiation.

Acknowledgement

The work was supported by the National Science Foundation of China (Grant No. 10871014).
References