Obtaining 2.nd Order Derivatives Using a Mixed Forward- and Backward Automatic Differentiation Strategy for use in Interval Optimization

This poster is also available from WWW at
<http://www.imm.dtu.dk/documents/users/os/fadbad.html>

Ole Stauning, < os@imm.dtu.dk >,
Department of Mathematical Modelling
Technical University of Denmark
2800 Lyngby

1 Introduction

One of the most successful areas of interval analysis is global optimization. Many different variants of the traditional branch and bound technique initially introduced by Skelboe/Moore [8, 10] has been proposed. A common thing about most of these algorithms is that the user has to provide subroutines for calculating the object function $f : C^2(D, \mathbb{R})$, the gradient ∇f and the Hessian $\nabla^2 f$. We will in this poster investigate a method for obtaining these derivatives of f automatically by means of forward- and backward automatic differentiation.

Automatic differentiation is widely known to people in the interval community. It is a method which uses the "chain rule" for composite functions to build a triangular system of equations. This system of equations can be solved in two different ways, both ways using the computational graph and the temporary values of the function evaluation itself [1, 9]. The two methods for solving the system of equations are usually called the forward- and the backward methods. The two methods are structurally very different – giving them different properties.

The forward method is easy to understand as well as to program on a computer. The method computes derivatives alongside the evaluation of the function itself. When the function evaluation has completed, the derivatives have also been calculated.

The backward method is based on a not as obvious way of using the "chain rule". The first step of the method is to evaluate the function itself while storing information about intermediate values and the structure of the computational graph for the evaluation. This first step is also called a "recording" of the function evaluation. When the evaluation is done the derivatives are calculated by traversing the "recording" in reverse order, propagating derivatives down to the initial values. Because of the "recording" and the reverse propagation this method is also the most difficult to program on a computer.

2 The FADBAD package

FADBAD is a C++ program package for doing forward as well as backward automatic differentiation of function evaluations programmed in C++. The package works by overloading all arithmetic operations used in the function evaluation by changing the type of the used variables to an automatic differentiation type which is generated by the FADBAD package. The automatic differentiation type can be generated on any arithmetic e.g. float, double, interval or even another automatic

differentiation type. This way forward- and backward automatic differentiation can be combined into delivering higher order derivatives.

The FADBAD package is public domain. The package and a technical report[1] can be obtained from the FADBAD homepage

<http://www.imm.dtu.dk/documents/users/os/fadbad.html>

or directly by ftp from

<ftp://ftp.uni-c.dk/uni-c/unicbe/FADBAD/FADBAD-1.0.tar.gz>.

3 Differentiating the object function

The number of operations used in differentiating a vector of functions $g : \mathbb{R}^n \to \mathbb{R}^m$ has been investigated [2, 4]. If $L(\cdots)$ is the number of operations used to calculate $\cdots$ then $L_{fwd}(g, J) = O(n) \cdot L(g)$ for the forward method and $L_{bwd}(g, J) = O(m) \cdot L(g)$ for the backward method, where J is the Jacobian matrix of g. An actual upper bound on the fraction $L(g, J)/L(g)$ can be found if we know what functions we will use.

Assume that we have a program which computes the function g and we use FADBAD to compute J. The cost of a single elementary arithmetic operation, e in forward automatic differentiation is

e	$L(e)$	$L_{fwd}(e)$
$x+y, x-y, -x$	$1A$	nA
$+x$	$1A$	0
$x*y, x/y$	$1M$	$2nM+nA$
$\mathrm{pow}(x,y)$	$1F$	$2nF+4nM+2nA$
$\mathrm{pow}(x,int), \cos(x)$	$1F$	$nF+2nM$
$\exp(x), \log(x)$	$1F$	nM
$\mathrm{sqrt}(x)$	$1F$	$2nM$
$\sin(x)$	$1F$	$nF+nM$
$\tan(x), \mathrm{atan}(x)$	$1F$	$nF+nM+nA$
$\mathrm{asin}(x)$	$1F$	$2nF+nM+nA$
$\mathrm{acos}(x)$	$1F$	$2nF+nM+2nA$

Where F is a function evaluation, M is a multiplication or a division and A is an addition or a subtraction. Taking the worst case $\mathrm{pow}(x, y)$ we obtain $L_{fwd}(g, J)/L(g) \leq 8n$.

In backward automatic differentiation we have

e	$L(e)$	$L_{bwd}(e)$
$x+y, x-y, +x, -x$	$1A$	$2mA$
$x*y, x/y$	$1M$	$2mM+2mA$
$\mathrm{pow}(x,y)$	$1F$	$2mF+4mM+2mA$
$\mathrm{pow}(x,int)$	$1F$	$mF+2mM+mA$
$\exp(x), \log(x), \sin(x), \cos(x)$	$1F$	$mM+mA$
$\mathrm{sqrt}(x)$	$1F$	$2mM+mA$
$\tan(x)$	$1F$	$mF+mM+mA$
$\mathrm{asin}(x), \mathrm{acos}(x)$	$1F$	$2mF+mM+2mA$
$\mathrm{atan}(x)$	$1F$	$mF+mM+2mA$

Also here the worst case is $\mathrm{pow}(x, y)$, and we obtain the bound $L_{bwd}(g, J)/L(g) \leq 8m$ for the backward method.[1]

[1] Note that the operation $\mathrm{pow}(x, y)$, where y is a variable, is not a common operation.

Obviously the independency of n in the backward method makes it in most cases superior to the forward method when calculating first order derivatives of an object function (since $m = 1$) with a high number of input variables n.

As the Hessian $\nabla^2 f : \mathbb{R}^n \to \mathbb{R}^n \times \mathbb{R}^n$ of f is found by differentiating ∇f, the number of operations to calculate f, ∇f and $\nabla^2 f$ becomes $L(f, \nabla f, \nabla^2 f) \leq 8n \cdot L(f, \nabla f)$ for the forward method as well as for the backward method. It is for this reason hard to say if we should use forward or backward automatic differentiation to calculate the Hessian. From the above list of operations we should expect the forward method to be a little better than the backward method.

4 Measuring the performance of automatic differentiation

Usually the computation time is used in measuring performance of algorithms. These measurements are however platform dependent. On multiuser systems two measurements on the same program can give completely different results, depending on the load of the system, making it difficult to compare performance. To circumvent these problems a C++ class `Iflop` has been made to overload the `INTERVAL` type defined in the BIAS/PROFIL package [7, 6]. This new class increments a flop-counter each time an arithmetic operation on the class is used. This approach – measuring Iflops (interval flops) instead of computing time – makes the measurements completely platform independent and independent of the load on multiuser systems. The flexibility of FADBAD makes it possible to apply automatic differentiation on these Iflops without problems.

A set of 50 test examples has been compiled. All examples are from the book "Global Optimization Using Interval Analysis" by E. Hansen [3] and from the technical report "A Global Minimization Method: The Multi-Dimensional Case" by C. Jansson and O. Knüppel [5]. The performance of obtaining first and second order derivatives using the FADBAD package has been measured on each example. First order derivatives are always computed using the backward method giving a measurement $LB = L_{bwd}(f, \nabla f)$. The second order derivatives are computed by differentiating the first order derivatives using the backward as well as the forward method giving the measurements $LBB = L_{bwd}(f, \nabla_{bwd} f, \nabla^2 f)$ and $LBF = L_{fwd}(f, \nabla_{bwd} f, \nabla^2 f)$ respectively.[2] The complexity of evaluating the function is $L = L(f)$. Furthermore we introduce the fractions LB/L, LBB/L and LBF/L.

The results are listed if figure 1. From this figure we see that $L_{bwd}(f, \nabla f)/L(f) \approx 4$, i.e. the cost of evaluating an object function and all partial derivatives is approximate 4 times as expensive as evaluating the function itself. Furthermore from LBB/L and LBF/L it can be seen that using the backward method to compute second order derivatives are generally a little less than twice as expensive as using the forward method. In figure 2 the values of LBF/L for the problems of dimension $n \leq 10$ has been plotted. From the figure we obtain $L_{fwd}(f, \nabla_{bwd} f, \nabla^2 f) \approx 5n \cdot L_{bwd}(f)$.

5 Conclusion

In this poster we have seen that higher order derivatives can be obtained by the use of automatic differentiation. Using the package FADBAD we are capable of obtaining second order derivatives in an efficient way by using the backward method to obtain the gradient while all calculations in the backward code again are differentiated using the forward method to obtain the Hessian. Using this mixed forward- and backward method we use approximate $5n$ times as many operations to compute $f, \nabla f$ and $\nabla^2 f$ as computing $f : \mathbb{R}^n \to \mathbb{R}$ alone. The way of computing derivatives presented in this report is especially useful in global interval optimization where the second order derivatives are used in a interval Newton iteration to prove existence of global minima.

[2] We use the notation $\nabla_{bwd} f$ to emphasize that backward differentiation was used to calculate first order derivatives.

Name of function f	n	L	LB	LBB	LBF	LB/L	LBB/L	LBF/L
Beale function [3]	2	16	64	392	232	4.00	24.50	14.50
Booth problem [3]	2	9	35	179	97	3.89	19.89	10.78
Box 3D function [3]	3	90	350	2624	1478	3.89	29.16	16.42
Branin [5]	2	11	44	214	128	4.00	19.45	11.64
Branin2 [5]	2	14	58	314	190	4.14	22.43	13.57
Chichinadze [5]	2	19	76	354	214	4.00	18.63	11.26
Exp2 [5]	2	120	480	2436	1434	4.00	20.30	11.95
Goldstein-Price [5]	2	22	92	532	308	4.18	24.18	14.00
Griewank 10 [5]	10	43	191	4591	2541	4.44	106.77	59.09
Griewank 2 [5]	2	10	42	222	140	4.20	22.20	14.00
Hartman 3 [5]	3	60	236	1790	971	3.93	29.83	16.18
Hartman 6 [5]	6	108	428	6272	3248	3.96	58.07	30.07
Kowalik problem [3]	4	110	440	4304	2356	4.00	39.13	21.42
Levy1 [3]	1	8	34	126	81	4.25	15.75	10.12
Levy10 [3]	5	57	227	2587	1377	3.98	45.39	24.16
Levy11 [3]	8	57	227	4003	2067	3.98	70.23	36.26
Levy12 [3]	10	57	227	4947	2527	3.98	86.79	44.33
Levy13 [3]	2	18	75	439	261	4.17	24.39	14.50
Levy14 [3]	3	26	108	900	510	4.15	34.62	19.62
Levy15 [3]	4	34	141	1525	841	4.15	44.85	24.74
Levy16 [3]	5	42	174	2314	1254	4.14	55.10	29.86
Levy18 [3]	7	58	240	4384	2326	4.14	75.59	40.10
Levy2 [3]	1	25	100	263	179	4.00	10.52	7.16
Levy3 [3]	2	51	205	1061	629	4.02	20.80	12.33
Levy3 [5]	2	51	205	1061	629	4.02	20.80	12.33
Levy4 [3]	4	34	141	1525	841	4.15	44.85	24.74
Levy5 [3]	2	57	227	1171	687	3.98	20.54	12.05
Levy5 [5]	2	57	227	1171	687	3.98	20.54	12.05
Levy8 [3]	3	33	134	1022	563	4.06	30.97	17.06
Levy9 [3]	4	57	227	2115	1147	3.98	37.11	20.12
Matyas problem [3]	2	7	31	159	91	4.43	22.71	13.00
Powell problem [3]	4	15	61	557	281	4.07	37.13	18.73
Price [5]	2	11	47	291	173	4.27	26.45	15.73
Rastrigin [5]	2	9	37	193	123	4.11	21.44	13.67
Rosenbrock function [3]	2	7	29	161	91	4.14	23.00	13.00
Rosenbrock [5]	2	7	29	161	91	4.14	23.00	13.00
Schwefel 1.2 [3]	50	1375	4225	426625	146675	3.07	310.27	106.67
Schwefel 1.2 [5]	4	18	62	550	234	3.44	30.56	13.00
Schwefel 3.1 [3]	3	21	81	651	357	3.86	31.00	17.00
Perturbed Schwefel 3.1 [3]	3	30	126	966	537	4.20	32.20	17.90
Schwefel 3.7 [3]	5	28	108	1378	718	3.86	49.21	25.64
Schwefel 3.7 [3]	10	63	243	5983	3013	3.86	94.97	47.83
Schwefel 3.7 [3]	30	203	783	56403	27693	3.86	277.85	136.42
Shekel 10 [5]	4	150	550	5822	3014	3.67	38.81	20.09
Shekel 5 [5]	4	75	275	2907	1499	3.67	38.76	19.99
Shekel 7 [5]	4	105	385	4073	2105	3.67	38.79	20.05
Simplified Rosenbrock [5]	2	7	29	161	91	4.14	23.00	13.00
Six hump camel back [5]	2	15	65	337	195	4.33	22.47	13.00
Three Hump Camel Back [5]	2	12	52	264	152	4.33	22.00	12.67
Treccani [5]	2	6	26	166	94	4.33	27.67	15.67

Figure 1: The complexity of differentiating using the FADBAD package on a scalar function $f : \mathbb{R}^n \to \mathbb{R}$. All measurements are in interval operations. $L = L(f)$ is the number of operations to evaluate the function. $LB = L_{bwd}(f, \nabla f)$ is the number of operations to evaluate the gradient using the backward method. $LBB = L_{bwd}(f, \nabla_{bwd} f, \nabla^2 f)$ is the number of operations to evaluate the gradient and the Hessian using the backward method. $LBF = L_{fwd}(f, \nabla_{bwd} f, \nabla^2 f)$ is the number of operations to evaluate the gradient using the backward method and the Hessian using the forward method.

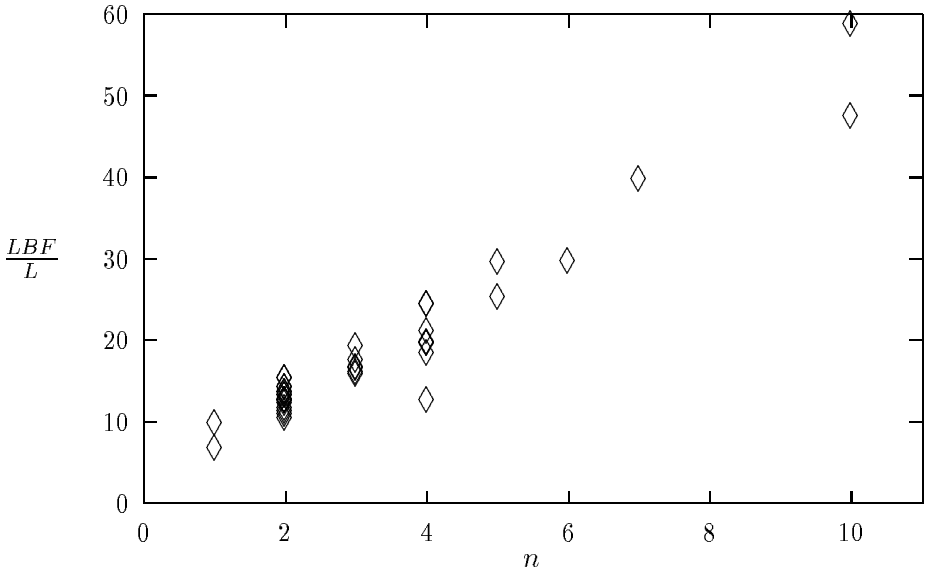

Figure 2: The complexity of evaluating f ∇f and $\nabla^2 f$ using forward-backward automatic differentiation class (LBF/L).

References

[1] Claus Bendtsen and Ole Stauning. FADBAD, a Flexible C++ Package for Automatic Differentiation. Technical Report IMM-REP-1996-17, Department of Mathematical Modelling, Technical University of Denmark, 1996.

[2] Hans-Christoph Fischer. *Schnelle automatische Differentiation, Einschließungsmethoden und Anwendungen.* PhD thesis, Fakultät für Mathematik, Universität Karlsruhe, Germany, 1990.

[3] Eldon Hansen. *Global Optimization Using Interval Analysis.* Marcel Dekker, inc., 270 Madison Avenue, New York., 1992.

[4] Masao Iri. History of automatic differentiation and rounding error estimation. In Andreas Griewank and George F. Corliss, editors, *Automatic Differentiation of Algorithms*, pages 3–16, 1991.

[5] C. Jansson and O. Knüppel. A Global Minimization Method: The Multidimensional Case. Technical report, Technische Informatik III, TU Hamburg-Harburg., 1992.

[6] Olaf Knüppel. BIAS – Basic Interval Arithmetic Subroutines. Technical report, Technische Informatik III, TU Hamburg-Harburg., 1993.

[7] Olaf Knüppel. PROFIL – Programmer's Runtime Optimized Fast Interval Library. Technical report, Technische Informatik III, TU Hamburg-Harburg., 1993.

[8] Ramon E. Moore. *Methods and Applications of Interval Analysis.* SIAM, Philadelphia, 1979.

[9] Dimitri Shiriaev. *Fast Automatic Differentiation for Vector Processors and Reduction of the Spatial Complexity in a Source Translation Environment.* PhD thesis, Fakultät für Mathematik, Universität Karlsruhe, Germany, 1993.

[10] Stig Skelboe. Computation of Rational Interval Functions. *BIT*, 14:87–95, 1974.