NARROWING OF INTERVALS BY PARTIAL DERIVATIVES

Eldar A.Musaev

Here is a review of methods to reduce a growth of intervals using an estimation by Taylor expansion. The comparison of these methods with traditional interval arithmetics, including the experimental results obtained by author, is made too.

СУЖЕНИЕ ИНТЕРВАЛОВ ПРИ ПОМОЩИ ЧАСТНЫХ ПРОИЗВОДНЫХ

Э.А.Мусаев

Ниже приводится обзор методов снижения роста интервалов при помощи разложения Тейлора. Также приведено сравнение этих методов с традиционной интервальной арифметикой, в том числе на полученных автором экспериментальных результатах.

It is supposed that a potential reader is familiar with the basic ideas of the interval analysis. If not, this information can be found, for example, in [1].

It is well known, that the main problem of interval computations is the fast growth of intervals. One of the reasons is hidden in the fact that

$$X - X = \{ x_1 - x_2 \mid x_1, x_2 \in X \}$$

instead of

$$X - X = \{ x - x \mid x \in X \}$$

There are many ways to decrease this effect and one of them is based on the idea of estimation by the first order derivatives, i.e.

$$F(X) \subseteq f(x_0) + F'(X)(X - x_0)$$

Any program is considered here as a function of input variables. Of course for every set of input values it will be continuous except division-by-zero cases. The values of partial derivatives are necessary here to estimate the result. The first way to compute them described by E.Hansen in [1] is a generalized interval arithmetic (g.i.a.). G.i.a. proposes to compute these derivatives simultaneously with the computation of f(x). The idea itself is very simple. Let us have an interval data type Seg and a generalized interval type

Type GSeg = Record v : Seg; d : [N]Seg End

where N is a number of input values, v is an estimation for the value, and d[i] is an estimation for the i-th partial derivative. It is evident that for i-th input value

$$\begin{array}{lccccccc} d = (& 0, & 0, \ldots, & 0, & 1, & 0, \ldots, & 0 &) \\ \text{index} = & 1 & 2 & i-1 & i & i+1 & N & \end{array}$$

Any numeric program consists of four arithmetic operations(without mentioning integer operations), combined by different operators. All these operations can be redefined in an accordance with elementary rules of differentiation, for example

$$A + B = (A.v+B.v, A.d[1]+B.d[1], \ldots, A.d[N]+B.d[N])$$

This method yields sufficiently good results, but increases computational complexity of arbitrary algorithm in proportion to the number of input variables. Moreover it increases both time and memory complexity. So computation of determinant of the n-th order matrix by Gauss method required $O(n^3)$ time and $O(n^2)$ memory in traditional arithmetic and $O(n^5)$ time and $O(n^4)$ memory in g.i.a. Taking into account the fact, that the approximation used in g.i.a. is not always better then traditional i.a., this cost is too high for the most of the practical tasks. But if an algorithm has few input variables then g.i.a. can be used independently of the number of output ones. Note also that traditional i.a. results are never better than g.i.a. ones, because g.i.a. includes traditional computation.

Another way to compute partial derivatives was proposed as an aposteriory interval analysis (a.i.a.) in [3] by Yu.Matijasevich. It was proposed to use a fast computations of partial derivatives described in [4] and [5] (in [4] this was used for a similar aim but without interval computations and guaranteed estimation). The main idea of fast computation of partial derivatives is not very complex. As it has been mentioned above, any program on any input data set could be represented as a sequence of simple assignments

$$x_{m+1} = x_{j_{m+1}}\ o_{m+1}\ x_{k_{m+1}}$$
$$\ldots$$
$$x_i = x_{j_i}\ o_i\ x_{k_i}$$
$$\ldots$$
$$y = x_n = x_{j_n}\ o_n\ x_{l_n}$$
where o_i is $+$, $-$, $*$ or $/$, for $\forall i=m+1..n$ $j_i,l_i<i$, and for $i\leq m$ x_i is an input variable. We need in values of $\partial y/\partial x_1, \ldots \partial y/\partial x_m$. Let us suppose that $\forall i$ $x_i = x_i(0)$, and $x_i(t)$ is an arbitrary C^1 function. Then
$$y'(x_1, \ldots x_m) = \partial y/\partial x_1\ x_1' + \ldots + \partial y/\partial x_m\ x_m'$$
but $x_i(t)$ is an arbitrary function, so the equation
$$d_1 x_1' + \ldots + d_m x_m' = y'$$
has the only decision $\partial y/\partial x_1$, ..., $\partial y/\partial x_m$. Now let us consider a more common task
$$d_1 x_1' + \ldots + d_k x_k' = y', \text{ where } m<k\leq n$$
But for $k=n$ there is an evident decision $(0,0,\ldots,0,0,1)$, because $y=x_n$. Let us suppose that we already have a decision for some k. How to compute decision for $k-1$? We know that
$$x_k = x_{j_k}\ o_k\ x_{l_k},\ j_k,l_k<k$$
so we can substitute x_k' by $(x_{j_k}\ o_k\ x_{l_k})'$ and get the other decision with zero coefficient at x_k', so it is the decision for $k-1$ task. Starting at the decision $(0,0,\ldots,0,1)$ for $k=n$ we can get by this way some decision for $k=m$. But such decision is unique, so that is the values of $\partial y/\partial x_1, \ldots \partial y/\partial x_m$.

The complexity of aposteriory interval analysis does not depend on the number of input variables, but depends on the number of output ones (increases proportionally). So the complexity of determinant computation by Gauss method is the same for traditional and aposteriory versions, but already that's not so for the matrix inversion.

Another problem of a.i.a. is a requirement to have all auxiliary results, because computation of derivatives could start only when the last value is computed. Moreover, cycles and reassignments in normal programs allow to increase the number of auxiliary variables so dramatically that computations may becomes impossible. Solution of this problem was found and described in [6]. All auxiliary results are obtained there during the inverse phase of the program, where auxiliary and input values are computed from output ones. An algorithm for such inversion of an arbitrary program is described in the same paper. Thus the source effectiveness of a.i.a. is achieved for the case of normal programs with possible cycles, conditionals, reassignments etc. The very idea of this inversion is in the extension of the output data set of a source program to make it invertible.

Sometimes the combination of g.i.a. and a.i.a. seems to be rational. Suppose we have many input values and compute a few auxiliary values, and later we compute a lot of output values from these auxiliary ones. In this case it is rational to compute these auxiliary values by a.i.a. (does not depend on the quantity of the input variables), and to finish computations by g.i.a. with auxiliary values as an input ones (does not depend on the quantity of the output values).

It is theoretically predicted that the traditional method would be better both by time and preciseness in the case of great source errors, while generalized arithmetic would be more precise in the case of small source errors. Aposteriory version will be nearer to generalized by preciseness and nearer to traditional by time.

Wishing to check these predictions and find the ways of realization of aposteriory interval system (originally it was aimed only for a linear program without reassignments) the author creates a system of arbitrary precision arithmetic, a special language for INterval COMputations and three compilers from INCOM to Turbo Pascal - traditional, generalized and aposteriory versions. These realization enables to make numerous numerical experiments. In particular such examples were tested as sums of rows, matrix operations (determinant, inversion), computations of integrals etc. Here are some results of testing on different examples (the relative error and time are shown for every test).

$\sum_{K=0}^{N} \frac{X^K}{K!}$, for N=10				$\sum_{K=1}^{N} \frac{(-1)^K X}{K}$, for N=10			
1000±1	T	95	0'06.0"	100±10	T	38	0'04.2"
	G	32	0'25.2"		G	8	0'10.7"
	A	35	0'29.0"		A	11	0'08.2"
$1000\pm1*10^{-8}$	T	10	0'06.0"	$1000\pm1*10^{-4}$	T	110000	0'04.0"
	G	03	0'25.3"		G	6458	0'11.1"
	A	03	0'29.4"		A	20000	0'08.2"

Determinant						A^{-1}xBxC					
100±1	1	1	T	8	0'03.7"	1	1	1	T	29	0'11.5"
1	100±1	1	G	7	0'45.0"	121±1	11	1	G	6	3'49.1"
1	1	100±1	A	7	0'22.5"	441±1	21	1	A	243	2'20.5"
100±1	100±1	100±1	T	22	0'03.7"	1	21	441±1	T	90000	0'07.4"
100±1	1	1	G	8	0'42.9"	1	11	121±1	G	1292	2'09.0"
1	100±1	1	A	22	0'18.4"	1	1	1	A	3494	1'11.2"
40401±1	10201±1	1	T	312	0'04.5"	1	1	1	T	6766	0'14.1"
201	101	1	G	303	1'04.4"	10201±1	101	1	G	8	4'09.8"
1	1	1	A	303	0'40.6"	40401±1	201	1	A	1188	2'25.5"
1	6	36±1	T	11	0'01.1"	1	201	40401±1	T	3668	0'08.6"
1	5	25±1	G	9	0'17.0"	1	101	10201±1	G	2	2'14.7"
1	1	1	A	42	0'08.3"	1	1	1	A	22	1'14.8"

$\int 4X^3-3X^2+2X-1\ \partial X + 1$, $X_N=1$, $X_0=0\pm100e-4$				$\int \sum(-1)^{I+1} X_I(3X_I-2)+1\partial X$, $X_I^0=1$, $X_I^0=0\pm100e-4$			
$X_0=$ 0±1e-4 N=20	T	923	0'13.0"	0±1e-4 N=20	T	2560	0'32.7"
	G	817	0'58.8"		G	1172	6'43.5"
	A	829	1'09.3"		A	1172	2'58.7"
N=10	T	2514	0'05.9"	N=10	T	6624	0'16.2"
	G	2628	0'36.7"		G	3976	3'20.4"
	A	2613	0'40.2"		A	3976	1'38.7"
N=200	T	1105	2'00.7"	N=200	T	4508	'5'30.1"
	G	728	12'22.4"		G	1128	1:11'02.3"
	A	747	11'30.6"		A	1128	25'98.0"

These data fully confirm the theoretical predictions and expected effectiveness of the a.i.a. The theoretical conditions for application both a.i.a. and g.i.a. are also confirmed. Both ways are really useful in the case of small input errors. In these experiments 'small' means about 1-3%. In the case of great number of input variables a.i.a. is preferable. In these experiments 'great' happens to be about 3-4 variables. Of course these number greatly depend on the realization. E.g. the slow realization of the external stack in a.i.a. will increase the number of variables when a.i.a. is better than g.i.a. Any change of precision will influence the permissible error level. Anyway, new generations of computers will make these ideas more practical and precious.

References

1.Moore R.E. Interval analysis. - Prentice Hall, 1966.
2.Hansen E. A generalized interval arithmetic. // Lect. Notes in Comp. Sci. - Springer-Verlag, 1975 - V.29 - P.7-18
3.Matijasevich Y. A posteriory interval analysis. // EUROCAL 85, Vol.2: Research Contribution. - Lect. Notes in Comp. Sci. - Springer-Verlag, 1985 - V.204 - P.328-334
4.Baur W., Strassen V. The complexity of partial derivatives. // Theoretical Computer Science - 1983 - V.22 - P.317-330
5.Linnainmaa S. Taylor expansion of the accumulated rounding error. // Bit - 1976 - V.16, N 2. - P.146-160
6.Musaev E. The inversion of arbitrary programs for the tasks of aposteriory interval analysis. // Proc. of the Soviet-Bulgarian seminar on the Numerical Processing, Pereslavl-Zalesski, USSR, 19-24 October 1987 - Inst.of the program systems,1989 - P.100-109 (in Russian)

USSR 191 011 Leningrad
Fontanka 27 LOMI AN USSR
(Mathematical institute)