East Asian Math J. **19** (2003), No. 1, pp. 17–26

ERROR ANALYSIS USING COMPUTER ALGEBRA SYSTEM

KEEHONG SONG

ABSTRACT This paper demonstrates the CAS technique of analyzing the nature and the structure of the numerical error for education and research purposes This also illustrates the CAS approach in experimenting with the numerical operations in an arbitrary computer number system and also in doing error analysis in a visual manner

1. Introduction

This paper starts with analyzing some of the classical examples of numerical errors from a CAS's perspective, in which the computers intrinsic imperfection displays itself in an intuitively appealing fashion Then as the main part of the paper, it discusses the various techniques for investigating the nature of numerical errors, which are feasible only with CAS. This paper shows the ways of constructing a model computer number system for investigating the fundamentals of computer error. This construction of the toy computer gives a perspective into the concrete methodology on how to experiment with the computer error. In doing so, CAS's symbolic manipulation and visualization makes it possible to perform algebraic operations in an arbitrary number system In this paper, we use *Mathematica* as

Received March 6, 2003 Revised May 2, 2003.

2000 Mathematics Subject Classification· 65G99

Key words and phrases. numerical error, Mathematica, computer number, machine epsilon, machine number

the choice for CAS for its symbolic, pattern-matching, and graphical programming language features ([3]).

2. Case Studies of Error-prone Circumstances

Case 1: If 0.1 is subtracted from 1 ten times consecutively, will the outcome be zero? If 0.1 is subtracted from 1 until we are supposed to get zero as the outcome, then we will get 1.38778×10^{-16} instead of zero, which should be surprising for the untrained. In order to reproduce this odd phenomenon, we can perform a set of *Mathematica* functions given below.

```
NestList [ # - 0.1 &, 1, 11]
NestWhileList [ # - 0.1 &, 1, Positive[#]&]
```

Similarly, although a simple explanation for this anomaly can be given, a few lines of *Mathematica* debugging codes should reveal much of the problem therein.

```
x = SetPrecision [.4, 16]; i=1;
While[Print[SetPrecision[x,16]] || (x!= 0.0 && i ≤ 4),
x= x -.1; i++]
0.4000000000000000
0.3000000000000000
0.2000000000000000
0.1000000000000000
2.775557561562891 × 10^-17
```

Case 2 ([1]): If the order of addition of real numbers is rearranged, then will the result be the same? In order to answer this question, consider two vectors, α and β, expressed in the *Mathematica* syntax below.

```
α = { 2.718281828, -3.141492654, 1.414213562,
    0.5772156649, 0.3010299957 };
β = { 1486.2497, 878366.9879, -22.37492, 4774714.647,
    0.000185049 };
```

Then the dot product of the vectors yields the result in the next line.

$\{4040.05, -2.75938\times10^6, -31.6429, 2.75604\times10^6, 0.0000557053\}$.

Now the anomaly is going to be visible as the order of addition changes from left to right and then right to left.

```
SetPrecision [Fold [Plus, 0, prod], 16]
SetPrecision [Fold [Plus, 0, Reverse[prod]], 16]
```

Now the importance of sorting operation as an error-reducing measure can be demonstrated as we perform the following *Mathematica* functions and then compare the results.

```
SetPrecision [Fold [Plus, 0, Sort[prod]], 16]
SetPrecision [Fold [Plus,0,Sort[prod, #1 > #2&]],16]
```

Case 3: Will the graphs of the following expression, $x^3-3x^2+3x-1 = x((x-3)x+3)-1$, $(x-1)^3$ look the same? Although the answer would be positive when the plotting is done on a reasonably wide interval, when the plot is drawn on a narrow range, the graphical outcome of those equations becomes disturbingly different.

```
Plot [ x^3 - 3x^2 + 3x -1, { x, 0.99998, 1.00002 }]
```

The fully expanded form shown above can be simplified somewhat using Horner's method ([1], [2]). Now we see that a small change in the form makes a considerable difference in the graphical outcome.

```
<< Algebra 'Horner'
parenthesized = Horner[x^3 - 3x^2 + 3x -1]
Plot[parenthesized, {x, 0.99998, 1.00002 }]
```

The optimal error-reducing form would be the factorized one, when possible, as demonstrated below in a visual manner.

```
Plot[ (x-1)^3, {x, 0.99998, 1.00002 }, PlotRange → All]
```

Now a visual rationale for the difference in look is possible with CAS. Let's take a peep inside the inner workings of the given computer number system using CAS, which help determine the desirable form in a visual and convincing manner.

```
<< NumericalMath'
MicroscopicError[ x^3 - 3x^2 +3x -1, { x, 0.999}]
MicroscopicError[ x((x-3)x +3) -1, { x, .999}]
MicroscopicError[ (x-3)^3, { x, .999}]
```

Now to exaggerate the discrepancy even further, we do the similar operation for the polynomial of higher degree.

```
f[x_] = Product[x-i, {i, 0, 30}]
g[x_] = Expand[f[x]]
```

As expected, we get more dramatic difference in graphical display for the algebraically identical polynomials.

```
pic1 = Plot[f[x], {x, 10, 20}]
pic2 = Plot[g[x], {x, 10, 20}, PlotPoints → 3]
```

Case 4: Is it possible that the variance of a dataset with three different data value be 0?

The variance of a dataset of different data is supposed to be greater than zero. However, under certain circumstance, it is possible to get zero or below for the variance due to numerical errors. The ordinary definition of the variance usually takes two different forms. Namely, they are

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 = \frac{1}{n}(\sum_{i=1}^{n} {y_i}^2 - n\bar{y}^2), \ \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i.$$

However, as far as the numerical errors are concerned, the quality of the two algebraically identical formulae for the variance of a dataset differs considerably. As shown in previous examples the former would have the numerical advantage as it has fewer operations. Here the difference between them can effectively demontrated using the CAS's symbolic manipulation capability. To demonstrate the difference of the two formulae, lets assume we have a dataset of size three.

$$\text{avg} = \frac{1}{3}(a+b+c);$$
$$\sigma_1 = a^2+b^2+c^2-3\text{avg}^2;$$
$$\sigma_2 = (a-\ \text{avg})^2+(b-\ \text{avg})^2+(c-\ \text{avg})^2;$$

However, these two algebraically identical expression yield different results numerically. To dramatize the discrepancy, we use a simulated number system where the seven-digit precision operations are forced.

```
SetArithmetic[7, MixedMode → True]
```

We need to put the header 'ComputerNumber' to ensure that all the subsequent numerical operations be done in the limited precision as intended.

```
{a, b, c} = ComputerNumber /@{ 10000, 10005, 10010 }
```

Now we can easily check the anomaly in which the computed value differs depending upon the form and, even worse, the variance becomes zero

$$a^2+b^2+c^2-\frac{(a+b+c)^2}{3}$$

0

$$\left(a-\frac{a+b+c}{3}\right)^2+\left(b-\frac{a+b+c}{3}\right)^2+\left(c-\frac{a+b+c}{3}\right)^2$$

50 0000000000000000

Case 5 ([2]): Can the following algebraically identical different expressions $\frac{x}{1+x}, \frac{1}{1+\frac{1}{x}}$, take different value when evaluated at a certain x?

The *Mathematica* function **Hold** is supposed to defer the execution of an expression until **ReleaseHold** is performed.

$$f[x\_] := \frac{x}{x+1}$$

$$g[x\_] := \texttt{Hold}\ [\frac{1}{1+\frac{1}{x}}]$$

Now we can see that the later expression gives the tighter interval, which implies the smaller error margin as demonstrated below.

$f[\texttt{Interval}\ [\{1,2\}]]$

$\texttt{Interval}\ [\{\frac{1}{3}, 1\}]$

$\texttt{ReleaseHold[g(Interval[}\{1,2\}\texttt{]}$

$\texttt{Interval}[\{\frac{1}{2}, \frac{2}{3}\}]$

3. Model Computer Number System and Machine Epsilon

In order to fully understand the basic nature of the numerical error, we get to construct a toy computer, which would indicate the inherent problems of the computer of any dimension. As a simple model serving our purpose effectively, we will construct a toy computer with the number system, $\pm(b_1 b_2 b_3)_2 \times 2^e$, $b_i = \{0,1\}, -2 \leq e \leq 1$, and visualize the discrete structure in a linear representation. Once displayed, what is visually outstanding is going to be the uneven distribution with a concentration in the middle of the scale. Now *Mathematica*'s symbolic capability plays a pivotal role in constructing the model computer structure that we intended to visualize. The variable 'floatnums' contains all the computer numbers of the system the toy computer represents.

```
floatnums = Distribute[{
   {" - 2^", "2^"},
   {"0", "1"},
   {"0", "1"},
  {"0","1"},
  {"2^"},
  {"-2","-1","0","1"}},
  List, List, List, StringJoin[##]&]
```

Next we need to convert the information in a string form into the numeric information.

```
Sort@ToExpression[floatnums]
```

Now the numeric data is prepared to be a primitive form so as to be processed in the *Mathematica* graphics function.

```
pts = Point [{#,0}]& /@%;

marks = Join[Table [Text[i,{i,0},{0,1}],{i,-3,3}],
  { Text[-.5,{-.5,0},{0,1}], Text[.5,{.5,0}, {0,1}]}];

toyComputer = Append[pts, marks];

Show[Graphics[toyComputer], PlotRange → { All, All },
  AspectRatio → 0.1, Prolog → PointSize[0.01]]
```

We can make the several steps listed above into a single procedure.

```
displayOnTheLine[xvals_] := Module[ {n = Length[xvals],
       pts, marks, i, floatScale},
       pts = Point[{#,0}]& /@ xvals;
       marks = {Text[xvals[[1]],{xvals[[1]],0}, {0,1}],
  Text[xvals [[-1]], {xvals[[-1]], 0 }, {0, 1}],
       Text[0,{0,0},{0 1}]};
       floatScale = Append[pts, marks];
       Show[Graphics[floatScale], PlotRange → All,
   AspectRatio → 0.1, Prolog → PointSize[.3/n]]
   ]
```

The process explained so far can further be simplified using the standard *Mathematica* package. First, we need to force the future computer arithmetic as 3-digit mantissa in a binary system for the experimentation purposes.

```
SetArithmetic[3, 2]
```

Then we need to generate the positive and negative numbers in the system and then sort them to display on a linear scale.

```
posNums = Table[Normal[ComputerNumber [1, i, j]],
          {j, -2, 1}, {i, 4, 7}]
negNums = Table[Normal[ComputerNumber [-1, i, j]],
          {j, -2, 1}, {i, 4, 7}]
Flatten[Union[posNums, negNums]]
machineNums = Sort[N[%]]
displayOnTheLine[machineNums]
```

Now the construction of the toy computer begs the discussion of the machine epsilon, the unit measure of error. The machine epsilon, the distance from a machine number 1 to the next machine number, which differs depending upon the given hardware and software system. In general, the distance from a given floating number to the next one is about R \$MachineEpsilon, a system variable for the machine epsilon in *Mathematica*. Although the machine epsilon is a built-in variable in *Mathematica*, it can be easily obtained for the given computer system using a procedure similar to the one given below.

```
MantissaExponent[ $MachineEpsilon]
```

$2.^{-52}$

```
g=1; ex = g = g/2;
While [ex> 0,
  g = 0.5 g;
  ex = g 0.98 + 1;
 ex = ex -1;
  If [ex >0, eps = ex]];
eps
```

This routine leads to another useful routine that determines the next number in a number system of a given computer system.

```
nextnum[x_] :  =
Module [{g= x}, ex = g = g/2;
 While[ex >0, g = 0.5g;
  ex = g 0.98 + x; ex=ex-x;
  If[ex >0, eps = ex]];
  eps ]
```

Now we generate the next number for a sequence from 1 to 100 and find an interesting pattern such that the distance to the next number for the integer 2 and 3 is 2 $MachineEpsilon and for 4, 5, 6, 7 is 4 $MachineEpsilon.

```
Table[nextnum[n] / $MachineEpsilon, {n, 1, 100}]
```

The difference between two consecutive machine numbers is called an *ulp* (unit in the last place, *i.e.*, one digit in the least significant place), which can be obtained using the next procedure.

```
uulp[x_] :=
    Module [{ t = Abs[N[x]], u },
    If [t < $MinMachineNumber,
      $MachineNumber,
      u = N[2^Floor[Log[2, t $MachineEpsilon ] -.5]];
      t=t - Release[t-u];
     If [ t = 0.  || t== 2u, 2u, u]
      ]
  ];
```

The size of an ulp varies depending on where the given number is located in the scale of machine numbers. For example, between 1 and 2 an ulp is equal to $MachineEpsilon and between 2 and 4 it is equal to 2 $MachineEpsilon. Consequently, we can guess that the distance between R and the next number to *R* is approximately *R* $MachineEpsilon.

```
Table [Ulp[n(1 + $MachineEpsilon)] / $MachineEpsilon, { n, 1, 100}]
```

This provides another algorithm that further simplifies the one given above.

```
Table [ Ulp[n + n $MachineEpsilon)], { n, 1, 10}]
```

In order to find the unit for the machine number to the left of 1, we can perform the next operation.

```
Ulp[1(1-$MachineEpsilon)]
```

Here is another example of the advantage of using CAS as the analyzing and visualizing tool. It displays the error pattern and size in the neighborhood of a certain number in the unit of ulp. The advantage of this visual approach is clear as it can often replace the computational one.

```
Microscope[Log[x], {x, 5, 5}]
MicroscopicError[Log[x], {x, 5, 10}]
```

Likewise, the virtue of the routine error-reducing technique shown below can be ascertained with the visualization technique using CAS.

$$\texttt{MachineError}\left[\ \sqrt{x^2+1}\ \texttt{-1, x} \to \texttt{0.00001}\right]$$

$$\texttt{MachineError}\left[\ \frac{x^2}{\sqrt{x^2+1}+1},\ \texttt{x} \to \texttt{0.00001}\right]$$

Now we are convinced that CAS makes a simple eyeball observation an effective and reliable error analysis.

REFERENCES

[1] W. Cheney and D. Kincaid, *Numerical Analysis, 2ed*, Brooks /Cole, 1996
[2] J. Keiper and R Skeel, *Elementary Numerical Computing with Mathematica*, McGraw-Hill, 1993
[3] S. Wolfram, *Mathematica A System for Doing Mathematics by Computer*, Addison-Wesley, 1991.

Department of Mathematics Education
Pusan National University
608-735 Pusan, Korea
E-mail: khsong@pusan.ac.kr
URL: www.mathematica. co.kr