Rational Arithmetic

Winfried Gerum

by Winfried Gerum

Arithmetic on computers can be surprisingly unreliable. The remedy is using rational arithmetic. To avoid cumbersome function calls, M operators are overloaded to support rational arithmetic.

These machines handle sound, graphics, text, and occasionally some arithmetic. And despite the wide array of uses, we still call them computers. Most of us probably think that computing is what these devices excel in.

Contrary to popular belief, these computers can be awfully bad at computing. Even if the processor does not have some unintended features (does anyone remember the flawed Pentium processors?). As long as we do integer arithmetic with small values there is no problem: all values are exact. When it comes to large values or to non-integer values, your system may be good for some surprises. For example, with MSM 3.0.8 you get:

```
> WRITE 36028797018963968*2
; WRITE (2**55)*2
72057594037927930
```

instead of: `72057594037927936`

and even:

```
> SET X=2**266  WRITE X*2=X
1
```

instead of: `0`

A power of two cannot yield a value ending with zero, and doubling a non-zero value should really give you a different value. The results of these computations are wrong! But it seems to be widely accepted behavior. What is behind it? Obviously the internal representation in MSM is a floating point number that has a limited number of bits to represent mantissa and exponent.

These numbers are well above the U.S. national debt (approx. 2**49 cents). So if no one cares about the national debt, why worry about much bigger numbers; unless, of course, you are a number scientist, an astronomer, or a physicist.

Then consider computations as simple as (1/N)*N with N being an integer value. Obviously the result should always be one. But that's not true on computers! Computing this expression on MSM (3.0.8) for integers N from 1 through 1,000,000, I got the correct value only 100 times.

Result(1/N)*N	# of Occurrences
.9999999999999998	6603
.9999999999999999	993297
1	100

Note that not a single result is above one. So in this case, over a long series of computations, the errors will not cancel each other out. Too much fuss about such a small difference? You don't have to be a chaos theorist to see the significance that very small changes can effect. It makes all the difference whether the determinant of a system of linear equations is zero or not. A non-zero value indicates that a solution exists. A zero value tells you that there is no solution. Therefore, a small value for the determinant should leave you in doubt, whether the solution is real or bogus!

There are some very common M functions and operators that are extremely sensitive to small deviations. Among them are the comparisons: numbers that are "almost equal" are in fact different:

```
> SET
X=".9999999999999998",Y=1
> WRITE A<Y
1
> WRITE A=Y
0
```

The Boolean interpretation of a string treats a value that is "almost zero" as different from zero (TRUE instead of FALSE). Many M operations use the integer interpretations on some of their arguments. The following operation discards all digits to the right of the decimal point! So "5.9999999999999998" becomes five, not six.

```
> SET X=5.9999999999999998,Y=6
> WRITE "X => ",$L($J("",X))
X => 5
> WRITE "Y => ",$L($J("",Y))
Y => 6
```

Is there a remedy short of going back to pencil and paper? Yes, because pencil and paper methods can be done reliably with. . . a computer.

Since old FORTRAN times we have called non-integer numbers REAL. But they are not at all what a mathematician calls a real number. REAL is meant to be an approximation to real numbers. But they do not even qualify as rational numbers (the set of all fractions of integer numbers). The REAL numbers are just a subset of rational numbers: they are decimal or binary fractions with a limited number of decimal (or binary) digits.

The problem comes with using the decimal system (or an equivalent) to represent numbers. A simple fraction like 1/3 cannot be represented with a finite number of decimal (or binary) digits. Whatever number of digits you choose, it has to be finite. Therefore the decision to use the decimal (or binary) system forces you to cut off at some point. This cutoff process introduces an error. Adding and multiplying floating point numbers may produce more digits (bits) than a word can hold, forcing again a truncation or rounding process.

If you accumulate enough small quantities, you can get any large quantity. A few years ago the Toronto stock exchange was computerized. They put up a display with their index. With each relevant transaction the index was updated in a continuous fashion. Over the course of some months, the index display indicated a severe bias — contrary to the general mood of the exchange. A recalculation of the index revealed that the display became increasingly out of sync with the original definition of the index because of a systematic loss with each computation.

Repeating a calculation with double precision or using a system with interval arithmetic may give you an idea of just how wrong a result is. But it is no real cure.

A way out of the dilemma is to compute with fractions of (arbitrary length) integers. Then operations using addition, subtraction, multiplication or division will not introduce an error. A number is represented by a pair of integers called counter and denominator. Addition of two fractions goes like this:

$$\frac{c_1}{d_1} + \frac{c_2}{d_2} = \frac{c_1 d_2 + c_2 d_1}{d_1 d_2}$$

The fractions can be normalized by allowing a minus sign on the counter only and by dividing them by their greatest common divisor. Normalizing them helps to avoid very large numbers as counter or denominator.

Calculation of the greatest common divisor:

```
   ;greatest common divisor of
   ;two integers
GCD(A,B)  ;
   N C
   S:$E(A)="-" $E(A)=""
   S:$E(B)="-" $E(B)=""
   F  S:A>B C=A,A=B,B=C Q:'A  D
   .S B=B#A
   Q B
```

When doing complex calculations in this way, if you get intermediate integers with more than fifteen digits, you should not trust the built-in math capabilities of your system. However, it is possible to do arithmetic in a pencil and paper fashion up to the string length limit of your system. For example, "arbitrary" length addition:

```
 ;add two arbitrary length integers
add(A,B) N C,I,R,SA,SB,Y
 S R=A,SA=$E(R)="-" S:SA $E(R)=""
 S Y=B,SB=$E(Y)="-" S:SB $E(Y)=""
 S I=$S($L(R)>$L(Y):$L(R),1:$L(Y))
 S R=$J(R,I),Y=$J(Y,I),C=0
 S:SA>SB R=$IR(R," 0123456789",
         99876543210),C=1
 S:SA<SB Y=$IR(Y," 0123456789",
         99876543210),C=1
 F I=I:-1:1 D
 .S C=C+$E(R,I)+$E(Y,I)
 .S $E(R,I)=$E(C,$L(C)),$E(C,$L(C))=
         ""
 D:SA-SB
 .I C S C=""
 .E  S R=IR(R,"0123456789",
         9876543210),SA=1,SB=1
 .F I=1:1:$L(R) I $E(R,I) S $E(R,1,I-
1)="" Q
 .I I=$L(R),R?1."0" S R=0
 .Q:SA'=SB  S C=1
 .F I=$L(R):-1:1 S C=C+$E(R,I),$E(R,I)=
         C#10, C=C\10 Q:'C
 S:C R=C_R
 I SA,SB S R="-"_R
 Q R
```

Routines for multiplication, integer division, and modulo can be written in a similar way.

Then you can combine all these functions into a library of subroutines. But you probably will never use them. Even with expressions of modest complexity it is very cumbersome to write down all these subroutine calls. Reading such a

program is as difficult as writing it. It would be nice to write down expressions the usual way and let the usual operators perform this rational arithmetic magic! The language standard does not prohibit your implementor from providing such clean math capabilities. If you can't convince them with money, logic or arm twisting, sit down and write your own expression evaluator in a few lines of M code. Then instead of writing:

```
SET X=$$ADD($$MULTIPLY
        ($$ADD(A,B),D),"3/7")
```

you write:

```
SET X=$$EXPR("(A+B)*D+(3/7)")
```

Typed programming languages provide at least some operator overloading. But usually it just means that +, -, *, and / can be applied to INTEGER, REAL, DOUBLE, and possibly COMPLEX numbers. Few programming languages (e.g. ADA) provide a mechanism for user-defined operator overloading.

This expression evaluator makes operator overloading available in M. It can easily be modified to overload the M operators to do complex math operations. Note that M indirection is essential to the working of this algorithm. All M variables except a few (?1" %"1U used by these functions internally) can be used in these expressions.

But let us first have a look at an example with a system of linear equations:

$$332x_1 + 301x_2 + 1157x_3 + 106x_4 = 1$$
$$156x_1 + 1079x_2 + 35x_3 + 22x_4 = 1$$
$$239x_1 + 979x_2 + 817x_3 + 85x_4 = 1$$
$$343x_1 + -277x_3 + 715x_3 + 64x_4 = 1$$

```
 ;evaluate M expression of rational entities
EXPR(%X) N %A,%B,%C,%I,%J,%O,%Q,%R,%U S
(%B,%J,%O,%Q,%R)="",%U=1
 F %I=1:1:$L(%X)+1 S %C=$E(%X,%I) D
 .I %U F  Q:%C="+"!(%C="-")'!(%X="'") S %I=%I+1,%C=$E(%X,%I)
 .S:%C="""" %Q='%Q Q:%Q
 .S:%C="(" %B=%B+1 S:%C=")" %B=%B-1 Q:%B
 .I %C="'",="><&!?"[$E(%X,%I+1) S %C=%C_$E(%X,%I+1),%I=%I+1
 .I %C="*",$E(%X,%I+1)="*" S %C="**",%I=%I+1
 .S %U="+-**/_'='<'>'&'!'?"[%C Q: '%U
 .S %A=$E(%X,%J,%I-$L(%C))
 .S %R=$S(%R="":%A,1:$$OP(%R,%O,%A)),%O=%C,%J=%I+1
 Q:%O="" $$EXPRA(%R)
 Q $$OP(%R,%O,%A)
;evaluate M expr_atom
EXPRA(%X) Q:%X?1"("1.E1")" $$EXPR($E(%X,2,$L(%X)-1))
 Q:%X?.1"-".N1"/"1.N %X
 Q:%X?1"-"1.E  $$MUL($$EXPR($E(%X,2,$L(%X))),-1)
 Q:%X?1"+"1.E  $$MUL($$EXPR($E(%X,2,$L(%X))),1)
 Q:%C?1"'"1.E  '$$EXPRA($E(%X,2,$L(%X)))
 Q:$P(%X,"(")?1"%"1U "<cannot handle"_%X_">"
 Q:%X?.1"^"1A.AN @%X
 Q:%X?.1"^"1"%".AN @%X
 Q:%X?.1"^"1A.AN1"(".E1")" @%X
 Q:%X?.1"^"1"%".AN1"(".E1")" @%X
 Q:%X?1"$"1.E @%X  ;svn, ssvn, functions
 Q:%X?1""""1.E @%X  ;strings
 Q %X
 ;evaluate binary operators
OP(%A,%O,%B) S %A=$$EXPRA(%A),%B=$$EXPRA(%B)
 Q:%O="+"  $$ADD(%A,%B)
 Q:%O="-"  $$SUB(%A,%B)
 Q:%O="*"  $$MUL(%A,%B)
 Q:%O="/"  $$DIV(%A,%B)
 Q:%O="="  $$EQUAL(%A,%B)
 Q:%O="**"  $$POWER(%A,%B)
 Q:%O="<"  $$LESS(%A,%B)
 Q:%O=">"  $$GREATER(%A,%B)
 Q:%O="'="  '$$EQUAL(%A,%B)
 Q:%O="'<"  '$$LESS(%A,%B)
 Q:%O="'>"  '$$GREATER(%A,%B)
 Q:@($$R2CAN(%A)_%O_$$R2CAN(%B))
```

```
 ; Solve a system of linear equations A*x=b with dimension n
SOLVE(N,A,X,B) N I,J,JPIVOT,K,L,L0,L1,R,S,W,Z
 F I=1:1:N S R(I,I)=1,Z(I)=I
 F K=1:1:N D
 .F I=K:1:N D  S L(Z(I),K)=$$EXPR("A(Z(I),K)-S")
 ..S S=0 F J=1:1:K-1 S S=$$EXPR("L(Z(I),J)*R(J,K)+S")
 .S L0=-1
 .F J=K:1:N D
 ..S L1=$TR(L(Z(J),K),"-") S:$$EXPR("L1>L0") L0=L1,JPIVOT=J
 .S L0=Z(K),Z(K)=Z(JPIVOT),Z(JPIVOT)=L0
 .F I=K+1:1:N D  S R(K,I)=$$EXPR("A(Z(K),I)-S/L(Z(K),K)")
 ..S S=0 F J=1:1:K-1 S S=$$EXPR("L(Z(K),J)*R(J,I)+S")
 F K=1:1:N D  S W(K)=$$EXPR("B(Z(K))-S/L(Z(K),K)")
 .S S=0 F I=1:1:K-1 S S=$$EXPR("L(Z(K),I)*W(I)+S")
 F K=1:1:N D  S X(N-K+1)=$$EXPR("W(N-K+1)-S")
 .S S=0 F I=N-K+2:1:N S S=$$EXPR("R(N-K+1,I)*X(I)+S")
 Q
```

This small program solves the problem.

The values of the above system of linear equations produces the following "solution" on MSM (3.0.8):

X_1 = .0039800356210444868
X_2 = .003597140594873016
X_3 = -.000371344648308079
X_4 = 0

The solution looks reasonable. The program, rewritten to do the trick with rational arithmetic, reveals that the previous solution is bogus.

If a solution exists, the results of computations like this is usually not that bad if done in M. Doing the computations with floating point arithmetic will typically incur a loss of about 5 digits. If a system gives you 8 decimal digits, a loss of 5 digits is dramatic. M gives you 15 digits to start with. If you loose 5, the result in most cases is perfectly acceptable.

The more digits of "precision" your system has, the smaller your chances of experiencing an example of bad computer math. But that just makes you complacent. As long as we use floating point arithmetic, our number crunchers occasionally crunch digits into meaningless dust. But with rational arithmetic, you never get a mega-flop (pun intended!).

The basic math capabilities of M implementation basically have the same inherent flaws as you will find in any other programming environment. But the other features like string handling and indirection allow for an easy and convenient implementation of rational arithmetic.

M

Winfried Gerum is with Winner Software, GmbH in Röttenbach, Germany. You may contact him by phone at 011-49-9195-940022 or by fax at 011-49-9195-940030.
