Ｊによる多倍長実数演算

慶応義塾大学理工学部
竹内寿一郎

１．はじめに

今年のはじめに「Ｊでパズルを」というシリーズをスタートした。その第１回目に「循環Ｎ 倍数の問題」を取り上げた。これは「最後尾の数字を先頭にもってゆくと元の数のＮ 倍になる数を求めよ」という問題であった。2倍のときの正解は18桁の循環数、すなわち繰り返すことにより、18、36、54、　桁の数値が解となる。3倍のときは28桁、4倍では6桁、5倍では42桁、6倍では58桁の循環数などなど、これらの結果を確かめるには多倍長整数演算を行わなければならない。従来多倍長計算のソフトとしてUBASIC などが知られているが、多機能をもつＪ言語で多倍長整数演算ソフトが容易に作成できるであろうと思った。そこで取り組んだのが第２回目の「多倍長整数演算」である。

ところが多倍長整数の表示を何桁でも表示できるようにベクトル(リスト)としたため、負の数の表示にアンダーラインを用いねばならず、ベクトルの要素として無限大という奇妙な表記法をとらねばならなかった。例えばマイナス20は_ ２ ０と表わさざるを得なかった。また、実際に演算用の動詞をつくって計算させると100桁程度の掛け算で数秒を要し、スピードの面からも数値計算に耐えうる演算システムとは言えなかった。さらにこれらを実数に拡張することを考えるとどのように実数を表示するかに困り果ててってしまった。ベクトルの中間にピリオド(小数点)が使えないからである。

そんなとき、志村氏から多倍長の整数演算がＪで出来るというお話を聞いた。多数桁の整数の後ろにxを付与すると多倍長演算をしてくれるというのである。そこで試しに実行すると確かに多倍長の演算をしてくれることが分かった。これを利用すれば演算時間を飛躍的に短くすることが出来ることを期待して、整数のみならず、実数の多倍長演算に拡張することを試みた。

２．Ｊの整数演算について

始めはまさかと思っていたのだが、いつの頃からできるようになったのか、Ｊのフリーウェアであ るJFW 3.02で実行してみた。

```
   1x+6x
7x
   123456789012345678901234 5x+12345678901234567890x
12345802358023580235802 35x
   12345678901234567 89x*12345678901234567 89x
1524157875323883675019051998750190521x
```

循環６倍数では58桁の数値を６倍して、

```
   135593220338983050847457627118644067796610169491525423728x*6x
813559322033898305084745762711864406779661016949152542372 8x
```

となり、確かに６倍すると桁がずれていることがわかる。

```
   5x%7x
|domain error
|   5x    %7x
```

```
   144x%12x
12x
```
割り算は割り切れないとエラーとなり、割り切れる場合には割り算を行う。

ところで、最近のバージョンでは多倍長整数演算結果が異なることが分かった。ここでは同じフリーウェアになったJ405dを使って試してみる。

```
   1x+6x
7
```
実行結果は、後ろにxが付いていない。
```
   12345678901234567890123456789x+123456789012345678901x
12345802358023580235802358023
   123456789012345678901x*123456789012345678901x
15241578753238836750190519987501905210
```
循環6倍数では、
```
   135593220338983050847457627118644067796610169491525423728x*6x
813559322033898305084745762711864406779661016949152542372880x
```
となり、後ろにxが付かない。
```
   5x%7x
5r7
```
割り切れない場合には有理数(分数)表示になる。
```
   10688763811772469185054480399079420x%10821521025816186345584514444x
86578987654567890r87654321098
```
有理数表示では有り難いことに適当に約分してくれる。
```
   144x%12x
12
```
割り切れる場合は割り算を実行してくれる。

最近のバージョンでは多倍長整数演算の実行結果の後ろにはxが付かないことに注意しなければならない。そこで少し多倍長整数演算の性質を調べてみる。

```
   154345677899088766667778899998880+887655432344567778876656667
1.54434e30
```
いくら長くても後ろにxが付いていなければ多倍長演算をしてくれない。
```
   123456789876543234566778898998765+123456789012345678901x*123456789012345678901x
1.52428e36
```
後ろにxがついていない整数があると、このように精度がおちてしまう。
```
   z=.123456789012345678901x*123456789012345678901x
   z
15241578753238836750190519987501905210
```
zに代入しておけば多倍長で記憶しておいてくれる。
```
   z+123456789012345678901x*123456789012345678901x
30483157506477673500381039975003810420
```
zを使って多倍長計算をすることができる。
```
   z
15241578753238836750190519987501905210
```

```
   zz=."."":z
   zz
1.52416e36
```
上のように、一度文字に直して再び数値に変えると精度が落ちてしまう。
```
   zz=."."(":z),'x'
   zz
1524157875323883675019051998750190521
```
文字化したら必ず後ろにxを付けて数値化しなければ多倍長整数にならない。

以上のことから鑑みてここでの多倍長実数演算用の動詞は処理系に依存するので、フリーウェアであるJ405dの上で作成することにした。

３．実数の表示について

Jの多倍長計算はあくまでも整数の演算のみなので、実数つまり小数点表示は使えない。そこで実数を指数部と仮数部に分けて表示するいわゆる、浮動少数点方式の表示を採用することにした。

例えば次の割り算の結果を考える。
```
   796456876x%576457x
796456876r576457
```
という有理数表示になってしまう。そこでこれを割り切れるように工夫する。
適当に被除数に10のべき乗を掛けて、剰余を引き、割り算をすると
```
   ((796456876*10x^20)-576457x|796456876x*10x^20)%576457x
13816414337929802222889 1
```
という答えになる。真の解は、
```
   796456876%576457
1381.64
```
そこでこれを指数部と仮数部で表示すると、
```
4 13816414337929802222889 1
```
つまり
```
0.13816414337929802222889 10^4
```
と書くことになる。
ところが
```
4 13816414337929802222889 1
```
1つの数値がベクトル(リスト)になる。
これでは後々大変でもあるし本来、実数1つはスカラ(アトム)であるべきなので、
```
+---------------------------+
|4 13816414337929802222889 1x|
+---------------------------+
```
と表示することにする。そのための動詞 su を用意した。
```
   su=:<@(,&'x')@":
```
これに付随して符号を調べる sign、符号変換 minus絶対値 abs などの動詞も用意した。
```
   sign=:[:*[:{:[:".[:>]
   minus=:3 : 0
<(":qq),' ',(":-yy),'x'[('qq';'yy')=:".>y.
)
   abs=:3 : 0
<(":qq),' ',(":|yy),'x'[('qq';'yy')=:".>y.
)
```

使用例　：

```
   a=.su 4 13816414337929802222891x
   a
+---------------------------+
|4 13816414337929802222891x|
+---------------------------+
   b=.minus a
   b
+----------------------------+
|4 _13816414337929802222891x|
+----------------------------+
   abs b
+---------------------------+
|4 13816414337929802222891x|
+---------------------------+
   sign"0 a,b
1 _1
```

なお、桁数が10桁より多いときは末尾に必ずxをつけねばならない。

```
   su 3 123456789123456789123456789
+-------------+
|3 1.23457e26x|
+-------------+
   su 3 123456789123456789123456789x
+-----------------------------+
|3 123456789123456789123456789x|
+-----------------------------+
```

４．割り算について

割り算の動詞divは単項で逆数、2項で商が計算できるようにしてある。この動詞を作成するにあたっての留意点は、仮数部は被除数を適当に10の冪乗倍して除数を引き割り切れる数にすること、指数部は両引数の指数部の引き算で良いが被除数と除数の比が1以上のとき1を加えねばならないことにある。特に1以上という条件が厄介で、ちょうど1であることを判定するには全ての桁を比較しなければならないからである。逆数についてもしかりで、例えば、ちょうど1のとき、その逆数の仮数部は1となるが、指数部は自動的に計算させるとマイナス1になる。そのためそれに1を加え (仮数部の先頭に小数点があるとしているので、すでに10で割った表現となっているから)、さらにちょうど1であるから1を加え、_1＋1＋1で、結果は1ということにする。逆数の中でちょうど1であると判定している部分は、if.*/(’1’={.":y),’0’=}.":yである。2項動詞の割り算では除数と被除数の大小を比較し、大きくもなく、小さくもないときに、等しいかどうかの判定をすることにしている。

```
   div=:3 : 0
n=:$":|y[’p y’=:".>y.
if.*/(’1’={.":y),’0’=}.":y do.r=:1 else. r=:0 end.
```

```
zz=:$":z=:(x0-y|x0=.10x^NN+n-r)%y
<(":r+>:-p),’ ’,(":z),’x’
:
m=:$xx=:":|x[’q x’=:".>x.[n=:$yy=:":|y[’p y’=:".>y.
if. xx<&([:".(<./m,n)&{.)yy do. r=:0
elseif. xx>&([:".(<./m,n)&{.)yy  do. r=:1
elseif.(xx,((>./0,n-m)#’0’),’x’)=&([:".(>:>./m,n)&{.)yy,((>./0,m-n)#’0’),’x’do.
elseif. ’’ do. r=:0
end.
zz=:$":z=:(x0-y|x0=.x*10x^NN+1-m-n-r)%y
<(":r+q-p),’ ’,(":z),’x’
)
```

使用例：

```
   NN=:30
   a=:su 1 _523456543290871111l1x
   b=:su _1 52345654329087657689x
   a
+-------------------------+
|1 _523456543290871111l1x|
+-------------------------+
   b
+-------------------------+
|_1 52345654329087657689x|
+-------------------------+
   div a
+-----------------------------------+
|0 _19103782593167932816767496756 21x|
+-----------------------------------+
   b div a
+--------------------------------------+
|_2 _10000000000000104417072822085424x|
+--------------------------------------+
```

５．足し算、引き算について

最も苦労したのが和と差の演算である。乗除算においては桁数はいい加減であっても多くとってあれば実害はない。しかし加減算では実数を整数に直すとき、1桁でも間違えると命取りになる。指数部をそろえると多くの場合桁数の異なる整数となるので、当然はじめに指定した精度NNより多い桁数で加減算を行うことになる。

和の場合では桁上がりが生じた場合、指数部に1を加えねばならないが、これはは結果の桁数が元の両引数のうちの最大桁を超えたかどうかで判定される。差の場合は桁落ちが生じたとき、不足分に対して後にゼロを付加した結果を出すようにしている。　和と差は1つの動詞でつくることが出来るが、高速演算をもくろむためには、少々無駄ではあるが、分けて処理した方が良いと

判断した。なお、waは正の符号をもつ引数同士の和、saは左引数の方が大きい場合の正の符号をもつ引数間の差をとる動詞、add と sub は正負の符号を決めるための動詞である。

```
   wa=:4 : 0
m=:$xx=:":|x[('p';'x')=:".>x.[n=:$yy=:":|y[('q';'y')=:".>y.
if. 0<p-q do. x0=:xx,(((p-q)+>:NN-m)#'0'),'x'[y0=:[yy,((>:NN-n)#'0'),'x'
else. x0=:xx,((>:NN-m)#'0'),'x'[y0=:yy,(((q-p)+>:NN-n)#'0'),'x' end.
<(":((>./(<:$y0),<:$x0)~:$Z)+>./p,q),' ',(NN{.Z=:":(".x0)+".y0),'x'
)
   sa=:4 : 0
m=:$xx=:":|x[('p';'x')=:".>x.[n=:$yy=:":|y[('q';'y')=:".>y.
if. 0<p-q do. x0=:xx,(((p-q)+>:NN-m)#'0'),'x'[y0=:[yy,((>:NN-n)#'0'),'x'
else. x0=:xx,((>:NN-m)#'0'),'x'[y0=:yy,(((q-p)+>:NN-n)#'0'),'x' end.
ZZ=:<./(MN=:>./($x0),$y0),$Z=:":(".x0)-".y0
<(":(>./p,q)-<:MN-ZZ),' ',(NN{.Z,(MN-ZZ)#'0'),'x'
)
   add=:4 : 0
if. (x. *&sign y.)=1 do.
if. 1=sign x.do.x. wa y.else.minus x. wa y.end.
else.
if.(val abs x.)>val abs y.
do.z=:x. sa y.
if.1=sign y.do.z=:minus z end.
else. z=:y. sa x.
if.1=sign x.do.z=:minus z end.
end.end.
)
   sub=:4 : 0
if. (x. *&sign y.)=_1 do.
if. 1=sign x.do.x. wa y.else.minus x. wa y.end.
else.
if.(val abs x.)>val abs y.
do.z=:x. sa y.
if._1=sign x.do.z=:minus z end.
else. z=:y. sa x.
if._1=sign y.do.z=:minus z end.
end.end.
)
```

使用例：

```
   a
+------------------------+
|1 _5234565432908711111x|
+------------------------+
   b
```

```
+------------------------+
|_1 523456543290876576 89x|
+------------------------+
   a add b
+----------------------------------+
|1 _5182219778579623453411000000000x|
+----------------------------------+
   a sub b
+----------------------------------+
|1 _5286911087237798768789000000000x|
+----------------------------------+
   b sub a
+----------------------------------+
|1 5286911087237798768789000000000x|
+----------------------------------+
   c
+------------------------+
|1 _523456543290876576 89x|
+------------------------+
   a add c
+-------------------------------------+
|_13 _5465780000000000000000000000000x|
+-------------------------------------+
```

次の valはボックスで囲まれた実数を単精度の数値に変える動詞である。

```
   val=:3 : 0
((*s)*".'0.',":|s=.{:u)*10x^{.(u=.".>y.)
)
   (val a)+val c
_5.41789e_14
   9087111111x-9087657689x
_546578
```

６．掛け算について

積の動詞の作成にあたっては次のことだけ気を付ければ良い。m 桁と n 桁の整数の掛け算は桁上がりがなければ m+n-1 桁になり、それを超えたときは桁上がりが生じているとして指数部を調整する。また精度不足の数値に対しては後にゼロと x を付与して目標の精度をもつ数に直す。

seki は正の引数同士の積を定義して動詞、mult は符号も考慮した積の動詞である。

```
   seki=:4 : 0
m=:$xx=:":|x[('p';'x')=:".>x.[n=:$yy=:":|y[('q';'y')=:".>y.
MN=:$":Z=:(|x)*|y
<(":p+q-(MN<m+n)),' ',(NN{.(":Z),((>./0,>:NN-m+n)#'0')),'x'
)
   mult=:4 : 0
```

```
if. (x. *&sign y.)=1 do.x. seki y.else.minus x. seki y.end.
)
```

使用例：

```
  a
+------------------------+
|1 _523456543290871111l1x|
+------------------------+
   b
+------------------------+
|_1 52345654329087657689x|
+------------------------+
   a mult b
+----------------------------------+
|0 _27400675271403048274333017711 5x|
+----------------------------------+
   (a mult b)div a
+-----------------------------------+
|_1 523456543290876576889999999983x|
+-----------------------------------+
```

７．総合応用例

また、応用上必要と思われるこの演算システムでの動詞をいくつか定義しておく。

```
   iota1=:3 : 0
,"1&’0x’each z=:su0 "0 >:i.y.
)
   su0=:3 : 0
<(":#":y.),’ ’,":y.
)
   fact=:3 : 0
mult/iota1 y.
)
```

応用例：

```
   NN=:30
   su0 8
+---+
|1 8|
+---+
  iota1 8
+-----+-----+-----+-----+-----+-----+-----+-----+
|1 10x|1 20x|1 30x|1 40x|1 50x|1 60x|1 70x|1 80x|
+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
   fact 8
+--------------------------------+
|5 40320000000000000000000000000x|
+--------------------------------+
   add/(su 1 100),div"0 fact"0 >:i.8
+--------------------------------+
|1 271827876984126984126984126984x|
+--------------------------------+
   ":!.16+/1,%!>:i.8
2.71827876984127
   add/(su 1 100),div"0 fact"0 >:i.20
+--------------------------------+
|1 271828182845904523533978449066x|
+--------------------------------+
   ":!.16+/1,%!>:i.20
2.718281828459045
   add/(su 1 100),div"0 fact"0 >:i.30
+--------------------------------+
|1 271828182845904523536028747135x|
+--------------------------------+
   ":!.20+/1,%!>:i.30
2.7182818284590451
   NN=:50
   add/(su 1 100),div"0 fact"0 >:i.30
+--------------------------------------------------+
|1 271828182845904523536028747135266237222570213098 34x|
+--------------------------------------------------+
   add/(su 1 100),div"0 fact"0 >:i.40
+--------------------------------------------------+
|1 271828182845904523536028747135266249775724709369 99x|
+--------------------------------------------------+
   add/(su 1 100),div"0 fact"0 >:i.50
+--------------------------------------------------+
|1 271828182845904523536028747135266249775724709369 99x|
+--------------------------------------------------+
   NN=:80
   add/(su 1 100),div"0 fact"0 >:i.100
+--------------------------------------------------------------------------------
|1 271828182845904523536028747135266249775724709369995957496696762772407663035 3
+--------------------------------------------------------------------------------
```