Rにおけるさらに別の多倍長精度パッケージ

中間栄治 † 中野純司‡

†(株)COM-ONE

‡統計数理研究所

2016年度統計数理研究所共同研究集会
「データ解析環境Rの整備と利用」

概要

はじめに

Rにおける多倍長精度パッケージとしては gmp,Rmpfr が既にあるが

- 扱いやすさ(多倍長精度としての数字の扱い)
- 速度面
- 複素数の扱い

以上において満足出来ない部分を個人的には感じていた.
よってこれらを可能な限り解決出来るように試みている最中である.
ただ、有り体に言えば,mpfr,mpc,mpackをRから扱うラッパーパッケージに過ぎない.

はじめての多倍長演算

よくある間違い

```
> library(Rmpfr)
> ans <- tan(0.05*10*pi)            ; ans  # 倍精度

[1] 1.633124e+16

> ans <- tan(mpfr(0.05,100)*10*pi) ; ans  # 倍精度を多倍長で100bit

1 'mpfr' number of precision  100   bits
[1] -38514314192475114.134131490935886

> ## 目的としては...
> ans <- tan(mpfr('0.05',100)*mpfr(10,100)*Const('pi',100))
> ans

1 'mpfr' number of precision  100   bits
[1] 11794639761737256078832190295792
```

有効桁数以内の整数以外は倍精度からの誤差を含む

名も無い多倍長パッケージでの例1

関数mpで囲んで多倍長演算に変換

```
> library(Rmpenv)
> default_prec(180)                          # 精度を100bitに設定
[1] 180
>     ans<-tan(0.05*10*pi)    ; ans      # 倍精度での演算
[1] 1.633124e+16
> mp( ans<-tan(0.05*10*pi) )  ; ans      # 多倍長での演算
 [1] -3.456584241974792526622263418085027236538728564743385508e+54
> mp( ans<-tan(0.05*10*pi), .debug=TRUE)# どのような変換を行ったか
ans <- tan(mpreal("0.05") * mpreal("10") * Const("pi"))
```

式に集中しやすく，可読性が良い

名も無い多倍長パッケージでの例2

mpに解析木を渡せば多倍長の解析木に変換

```
> options(digits=22)
> let<-substitute( {a<-c((.1+.1i),(.1+.1i)); sqrt(a[1]^0.1+a[2]^0.1)})
> eval(let)

[1] 1.2814621799274523716860+0.0503487861756312243821 7i

> library(Rmpenv); invisible(default_prec(180))
> mp(let)

 [1] 1.2814621799274525656592046873706294074987378864158477 33e+00
     5.0348786175631217356561678732197746490046653789067268 29e-02i

> mp(let, .debug=T)

{
    a <- c((mpreal("0.1") + mpcplx("( 0 0.1 )")), (mpreal("0.1") +
        mpcplx("( 0 0.1 )")))
    sqrt(a[1]^mpreal("0.1") + a[2]^mpreal("0.1"))
}
```

関数は多倍長で定義されているものに限られる

Rmpfrと名も無いパッケージとの主な差異

- S4(Rmpfr) と S3(名も無いパッケージ)
- 精度はdefault_prec関数で設定(名も無いパッケージ)
- 複素数も扱える(名も無いパッケージ)
- 多分速い(名も無いパッケージ)

内積で比較(実数)

```
> library(Rmpfr)
> A<-mpfr(1:100000,100)
> system.time(a0<-sum(A*A)) ;a0

   user  system elapsed
  1.908   0.016   1.926

1 'mpfr' number of precision  180   bits
[1] 333338333350000

> library(Rmpenv); invisible(default_prec(100))
> A<-mpreal(1:100000)
> system.time(a1<-sum(A*A)) ;a1

   user  system elapsed
  0.228   0.000   0.225

 [1] 3.33338333350000000000000000000e+14

> system.time(a2<-dot(A,A)) ;a2  # mpackのL1BLAS Rdotを呼び出す

   user  system elapsed
  0.164   0.000   0.165

 [1] 3.33338333350000000000000000000e+14
```

内積で比較(複素数1)

```
> library(Rmpenv); invisible(default_prec(100))
> A<-mpcomplex(real=1:100000,imag=1:100000)
> system.time(a11<-sum(A*A))   ; a11

   user  system elapsed
  0.520   0.004   0.525

 [1] 0.00000000000000000000000000000000e+00
     6.66676666700000000000000000000000e+14i

> system.time(a12<-cdotu(A,A)) ; a12 # mpackのL1BLAS Cdotuを呼び出す

   user  system elapsed
  0.408   0.000   0.409

 [1] 0.00000000000000000000000000000000e+00
     6.66676666700000000000000000000000e+14i
```

内積で比較(複素数2)

```
> library(Rmpenv); invisible(default_prec(100))
> A<-mpcomplex(real=1:100000,imag=1:100000)
> system.time(a21<-sum(Conj(A)*A)) ; a21

   user  system elapsed
  0.648   0.008   0.656

 [1] 6.6667666670000000000000000000e+14
     0.0000000000000000000000000000e+00i

> system.time(a22<-cdotc(A,A))     ; a22 # mpackのL1BLAS Cdotcを呼び出す

   user  system elapsed
  0.424   0.000   0.424

 [1] 6.6667666670000000000000000000e+14
     0.0000000000000000000000000000e+00i
```

行列積で比較(実数)

```
> N<-50
> library(Rmpfr)
> A<-matrix(mpfr(runif(N^2),100),N,N)
> system.time(A%*%A)

   user  system elapsed
  5.844   0.000   5.852

> N<-50
> library(Rmpenv); invisible(default_prec(100))
> A<-matrix(mpreal(runif(N^2)),N,N)
> system.time(A%*%A) # mpackのL3BLAS Rgemmを呼び出す

   user  system elapsed
  0.056   0.000   0.055
```

crossprod, tcrossprodも実装(名も無いパッケージ)
行列積は$O(n^3)$の演算量のアルゴリズムなのでNが2倍なら8倍遅くなる

行列積で比較(複素数)

```
> N<-100
> library(Rmpenv); invisible(default_prec(100))
> A<-matrix(mpcomplex(real=runif(N^2),imag=runif(N^2)),N,N)
> system.time(A%*%A) # mpackのL3BLAS Cgemmを呼び出す
   user  system elapsed
   2.36    0.00    2.36
```

複素数版crossprod, tcrossprodも実装(名も無いパッケージ)
mpcomplexはRのbase::complex相当の関数
mpcplxというmpcのインターフェース仕様の関数もある

連立一次方程式を解く

```
> ## ================================ 通常の倍精度のsolve
> N<-13
> hilbert <- function(n) { i <- 1:n; 1 / outer(i - 1, i, "+") }
> h <- hilbert(N); try(sh<-solve(h))

Error in solve.default(h) :
   システムは数値的に特異です: 条件数の逆数 = 2.93894e-19

> ## ================================ 多倍長のsolve
> library(Rmpenv); invisible(default_prec(1000)) # 精度を1000に
> N<-128
> hilbertmp <- function(n) { i <- mpreal(1:n); 1 / outer(i - 1, i, "+") }
> h <- hilbertmp(N); sh <- solve(h)
> all.equal(round(as.double(sh %*% h), 10),diag(N)) # 単位行列が得られるか

[1] TRUE
```

複素数版solveも実装(名も無いパッケージ)
一方でroundは未実装なので一旦倍精度に変換

おわりに

- 扱いやすさについては, 未サポートの関数も含め今後改善を行う
- 速度面は通常の倍精度に比べれば遥かに遅いが, 通常の倍精度では行えない事が行える
- LAPACK(mpack)に含まれる行列の各種分解ルーチンは今後追加
- パッケージ名は...