I learnt from a recent post on [John Cook’s excellent blog](https://www.johndcook.com/blog/2020/03/18/gnu-mpfrr-wrapper/) that it’s really easy to do extended floating point computations in R using the Rmpfr package. Rmpfr is R’s wrapper around the C library MPFR, which stands for “**M**ultiple **P**recision **F**loating-point **R**eliable”.

The main function that users will interact with is the mpfr function: it converts numeric values into (typically) high-precision numbers, which can then be used for computation. The function’s first argument is the numeric value(s) to be converted, and the second argument, precBits, represents the maximal precision to be used in numbers of bits. For example, precBits = 53 corresponds to [double precision](https://en.wikipedia.org/wiki/Double-precision_floating-point_format).

In his blog post, Cook gives an example of computing to 100 decimal places by multiplying the arctangent of 1 by 4 (recall that , so ):

4 \* atan(mpfr(1, 333))

# 1 'mpfr' number of precision 333 bits

# [1] 3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706807

***Why does he set the precision to 333 bits?*** [This link](https://stackoverflow.com/questions/10484332/how-to-calculate-decimal-digits-of-precision-based-on-the-number-of-bits) suggests that with bbits, we get decimal digits of precision. (Reality for floating point numbers is not quite as straightforward as that: see [this](https://www.exploringbinary.com/decimal-precision-of-binary-floating-point-numbers/) for a discussion. But for our purposes, this approximation will do.) Hence, to get 100 decimal places, we need around bits, so he rounds it up to 333 bits.

The first argument to mpfr can be a vector as well:

mpfr(1:10, 5)

# 10 'mpfr' numbers of precision 5 bits

# [1] 1 2 3 4 5 6 7 8 9 10

As the next code snippet shows, R does NOT consider the output of a call to mpfr a numeric variable.

x <- sin(mpfr(1, 100))

x

# 1 'mpfr' number of precision 100 bits

# [1] 0.84147098480789650665250232163005

is.numeric(x)

# [1] FALSE

We can use the asNumeric function to convert it to a numeric:

y <- asNumeric(x)

y

# [1] 0.841471

is.numeric(y)

# [1] TRUE

***Can we use the more familiar* as.numeric *instead?*** According to the function’s documentation, as.numeric coerces to both “numeric” and to a vector, whereas asNumeric() should keep dim (and other) attributes. We can see this through a small example:

x <- mpfr(matrix(1:4, nrow = 2), 10)

x

# 'mpfrMatrix' of dim(.) = (2, 2) of precision 10 bits

# [,1] [,2]

# [1,] 1.0000 3.0000

# [2,] 2.0000 4.0000

asNumeric(x)

# [,1] [,2]

# [1,] 1 3

# [2,] 2 4

as.numeric(x)

# [1] 1 2 3 4