

## Online Appendix B

# A MATLAB Multiple Precision Package

## B.1 The Mulprec Package

In the following we describe the basics of **Mulprec**, a collection of MATLAB m-files for, in principle, unlimited multiple precision floating-point computation. The version of Mulprec presented here was worked out by the first author in 2001. It is a preliminary version so bugs may still exist.

Originally, a shorter version of this package and text was meant as motivation for a Master's project at the Royal Institute of Technology (KTH), Stockholm. Some new ideas about chopping strategies and error estimation and control have been applied in some of the m-files for the basic operations and elementary functions.

In the following we also give several examples of the possible use of the Mulprec package.

### B.1.1 Number Representation

A **normalized Mulprec number** is a row vector  $x$  with the usual MATLAB notations; the value of  $x$  reads

$$\text{val}(x) = P^{x(1)} \sum_{j=2:k} x(j) P^{k-j}, \quad P = 10^7, \quad k \geq 2.$$

The  $x(j)$ ,  $j > 1$ , are integers called **gytes** (or *gits*), i.e., giant digits. They should all have the same sign (equal to  $\text{sgn}(x)$ ), and

$$|x(j)| < P, \quad j = 2 : k, \quad x(2) \neq 0.$$

Thus, we have a position system with base  $P = 10^7$ , where  $x(1)$  is the exponent of  $\text{val}(x)$  in a floating-point representation with base  $P$ .

Please note that  $P^{x(1)}$  denotes the unit of the *least significant* gyte, contrary to the traditional floating-point convention.<sup>198</sup> The length  $k$  of a Mulprec number  $x$  may vary

<sup>198</sup>It seems to be rather easy to change this if desirable.

during a computation. As an example, with an absolute error less than  $P^{-10} = 10^{-70}$ ,  $\pi$  equals the following 12-gyte number.

```

-10 3 1415926 5358979 3238462 6433832 7950288
      4197169 3993751 0582097 4944592 3078164

```

The decimal point, or rather the *gyte point*, is located immediately after column  $12 - 10 = 2$ . Note that

$$1 = [0, 1], \quad 0.5 = [-1, 5000000], \quad -0.125 = [-1, -1250000].$$

We call the MATLAB numbers **floats**. You rarely have to write the Mulprec form of numbers that are *exactly representable as floats*. The commands for the elementary operations and most functions are constructed in such a way that they *accept single floats (not expressions) as input data* and convert them to normalized Mulprec numbers by means of the command *npr*, or *npc* for complex floats; see below. (Expressions with Mulprec operations are, however, allowed as input data.) Mulprec distinguishes between floats and Mulprec numbers by the length, which is equal to one or larger than one, respectively.

For a **complex** normalized Mulprec number, these conditions typically hold for both the real and the imaginary part. The exponent and the length are common for both parts with an exception:  $x(2)$  may thus be zero for one of the parts.

It is fundamental for Mulprec that  $P$  can be squared without overflow with some margin. In fact,

$$2^{53} > 90P^2.$$

Hence, if the shorter one of two positive normalized Mulprec numbers has at most 90 gytes, we can obtain their product by the multiplication of gytes and addition of integers so that the sums do not exceed  $2^{53}$ . Typically, there is only one normalization in a multiplication.

The normalized representation of  $x$  is *unique* (if  $x \neq 0$ ). For example, note that if you subtract two positive normalized Mulprec numbers, the gytes of the result may have varying signs, unless you **normalize** the result by the Mulprec operation *nize* (or the simpler operation *rnize* if the number is real). Since the operation *rnize* is not fast compared to the operations *add* and *sub*, there is as a rule no normalization in *add* and *sub*.

For such reasons we now introduce a more general concept: the **legal Mulprec number**.  $\text{val}(x)$  has the same value and the same form as the normalized Mulprec number, but all the  $x(j)$  need not have the same sign and they have a looser bound:<sup>199</sup>  $|x(j)| < 45 P^2$ . Evidently such a representation of a number is not unique.

Allowing this more general type of Mulprec number in additions and subtractions makes it *unnecessary to transport carry digits inside these operations*; this is typically done later if a normalization is needed.<sup>200</sup>

A typical suboperation of the normalization is to subtract a multiple  $cP$  from one of the  $x(j)$ ; this is typically compensated for by adding  $c$  to  $x(j - 1)$ , in order to keep  $\text{val}(x)$  constant. (Is that not how we learned to handle the carry in addition in elementary school?)

Multiplication, division, elementary functions, etc. do include normalization, both of the operands and of the results. Only normalized numbers should be printed.

<sup>199</sup>The addition of two legal numbers does not cause overflow, but the sum can be illegal at first and must be immediately normalized; see the next footnote.

<sup>200</sup>An exception: if the result of an add or a subtraction has become illegal, then it becomes acceptable after an automatic call of *nize* inside *add.m* (or *sub.m*).

## B.1.2 The Mulprec Function Library

About 60 m-files for different Mulprec operations and functions have been developed so far. The numbers in the beginning of the lines of the listings below are only for making references in the text more convenient. They are thus *not* to be used in the codes and your commands. Since the condensed comments in the table below may be unclear, you are advised to study the codes a little before you use the system.

Mulprec numbers are typically denoted  $x, y, z$ . As mentioned above, most of the commands also accept floats as input if it makes sense. In a command like  $z = \text{mul}(x, y, s)$ , the parameter  $s$  means the number of gytes wanted in the result (including the exponent; hence it equals the length in the MATLAB sense). It is optional; if  $s$  is omitted, the *exact* product is computed and normalized (not chopped).

An asterisk means that the code is longer than 500 bytes. The absence of an asterisk usually indicates, e.g., that the code is a relatively short combination of other library codes. The numbers at the beginning of the lines of the following tables are not used in the computations; they are just for easy reference to the table and to Mulprec.lib.

## B.1.3 Basic Arithmetic Operations

Addition and subtraction were commented on above. Two routines for subtraction are given; 2b is shorter, but slower than 2a. Multiplication is performed as in elementary school—the amount of work is approximately proportional to the product of the sizes of the factors. Perhaps one of the fast algorithms, presented in Knuth [2, Sec. 4.3.3] in the binary case, will be adapted to the gyte system in the future.

In the table below, the shorter of the operands in `mul.m` is chosen to be the multiplier. In order to avoid overflow (in the additions inside the multiplication), the multiplier is chopped to 90 gytes (at most 623 decimal places). There are bounds also for the accuracy for division, square root, elementary functions, etc., since multiplication is used in their codes.

$1/x$  and  $1/\text{sqrt}(x)$  are implemented by Newton's iteration method, with variable precision that (roughly) doubles the number of gytes in each iteration. The initial approximation is obtained by the ordinary MATLAB operations (giving approximately 16 correct decimals). See more details in the Mulprec library. The square root algorithm is division-free.

At present, some limitations of Mulprec are set by the restriction of the length of the shorter operand of a multiplication to at most 90 gytes. It does not seem to be very difficult to remove these bounds, or at least to widen them considerably.

No.	m-file	Function	Operation
1*	<code>add.m</code>	$z = \text{add}(x, y)$	$z = x + y$
2a*	<code>sub.m</code>	$z = \text{sub}(x, y)$	$z = x - y$
2b	<code>subb.m</code>	$z = \text{subb}(x, y)$	$z = x + (-y)$ ,
3*	<code>mul.m</code>	$z = \text{mul}(x, y, s)$	$z = x \cdot y$ , $s$ optional
4	<code>recip.m</code>	$z = \text{recip}(x, s)$	$z = 1/x$
5	<code>div.m</code>	$z = \text{div}(x, y, s)$	$z = x/y$
6	<code>mi.m</code>	$z = \text{mi}(x)$	$z = -x$
7*	<code>musqrt.m</code>	$[y, \text{iny}] = \text{musqrt}(x, s)$	returns $(\sqrt{x})^{\pm 1}$

### B.1.4 Special Mulprec Operations

The operation `chop.m` is more general than just chopping to a desired length; see the code in the Mulprec library. The normalization code `rnize` still has a bug that violates the uniqueness. It can happen, e.g., that the last two bytes of a positive number read, say, `-1 9999634`. Such nine-sequences may also occur at other places in the vector. Sometimes such a representation is more easily interpreted than a strictly normalized Mulprec number. I have therefore not yet tried to eliminate this bug.

No.	m-file	Function	Operation
8	<code>npr.m</code>	<code>xx = npr(x)</code>	converts real float to normalized Mulprec number
9	<code>npc.m</code>	<code>xx = npc(x)</code>	converts complex float to normalized Mulprec number
10	<code>flo.m</code>	<code>y = flo(x)</code>	approximates Mulprec number by float
11*	<code>chop.m</code>	<code>y = chop(x,s)</code>	returns approximately equivalent Mulprec number, length $s$
12*	<code>rnize.m</code>	<code>y = rnize(x)</code>	normalizes real Mulprec number
13	<code>nize.m</code>	<code>y = nize(x)</code>	normalizes complex Mulprec number
14	<code>elizer.m</code>	<code>y = elizer(x)</code>	eliminates zero bytes in Mulprec number, left and right
15	<code>muzero.m</code>	<code>y = muzero(x)</code>	if $x = 0$ , $y = 1$ , else $y = 0$

## B.2 Function and Vector Algorithms

### B.2.1 Elementary Functions

In the computation of  $e^x$ ,  $x$  real, we first seek  $\bar{x}$  and an integer  $n$  such that

$$e^x = e^{\bar{x}} P^n \quad \text{and} \quad |\bar{x}| < \frac{1}{2} \log P.$$

Then, for an appropriate integer  $m$ ,  $e^{\bar{x}/2^m}$  is computed by the  $k - 1$  term of the Maclaurin expansion, a Horner scheme with variable precision.  $e^{\bar{x}}$  is then obtained by squaring the sum of the Maclaurin expansion  $m$  times.

By a combination of heuristic theory and experiment it has been found that the volume of computation is proportional to  $m + ck$ , where  $c = 0.4$ . At each call, the code computes approximately optimal values of the parameters  $m$  and  $k$  from a formula for finding the minimum of  $m + ck$  with the constraint that the bound for the relative error of  $e^{\bar{x}/2^m}$ , due to the Maclaurin truncation and the squarings, does not exceed  $P^{-s}$ .

A similar idea is applied for  $e^{ix}$ . Now  $\bar{x} \in [-8\pi, 8\pi]$ , and we use  $k - 1$  terms of the Taylor expansion into powers of  $x/2^m$ . These methods are inspired from ideas developed by Napier and Briggs when they computed the first tables of logarithms; see Goldstine [1, Sec. 1.3].

The algorithms in `lnr.m` and `muat2.m` are based on Newton's method for the equations  $e^y = x$  and  $\tan y = x$ , respectively, with initial approximations from the MATLAB opera-

## B.2. Function and Vector Algorithms

B-5

tions  $\ln x$  and  $\text{atan2}(y, x)$ . The commands `muat2`, `lnc`, and `mulog` do not yet allow floats as input, and the codes are not well tested.

No.	m-file	Function	Operation
16*	<code>expo.m</code>	$y = \text{expo}(x, s)$	$y = e^x, x$ real
17*	<code>expi.m</code>	$[\text{cox}, \text{six}, \text{eix}] = \text{expi}(x, s)$	$\cos x$ and (optionally) $\sin x, e^{ix}, x$ real
18	<code>muexp.m</code>	$w = \text{muexp}(z, s)$	$y = e^z, z$ complex; not yet implemented
19*	<code>lnr.m</code>	$y = \text{lnr}(x, s)$	$y = \log z, x > 0$
20*	<code>muat2.m</code>	$v = \text{muat2}(y, x, s)$	adapted from $\text{atan2}(y, x)$ ; not yet with float input
21a	<code>lnc.m</code>	$w = \text{lnc}(z, s)$	$w = \log z, z \neq 0$ ; not yet with float input
21b	<code>mulog.m</code>	$w = \text{mulog}(z, s)$	a better name for 21a

### B.2.2 Mulprec Vector Algorithms

A **Mulprec column vector** is represented by a (MATLAB) rectangular matrix. A **Mulprec row vector** is a row of Mulprec numbers (where each Mulprec number is a row of bytes). In a **rectangular Mulprec matrix**, each column is a Mulprec column vector, and each row is a Mulprec row vector. Thus, we can say that a Mulprec matrix is a row of rectangular (MATLAB) matrices, all of the same size. The following set of operations is very preliminary. The function `fixcom.m` returns, if possible, a Mulprec number (or Mulprec vector) with the same fixed comma representation (i.e., same exponent and size) as the normalized Mulprec number  $a$ .

These vector functions were worked out for an application to repeated Richardson  $h^2$  extrapolation; see the m-file `rich3.m`.

No.	m-file	Function	Operation
30*	<code>fixcom.m</code>	$y = \text{fixcom}(x, a)$	$x, y$ Mulprec vectors; returns $y \approx x, y(1) = a(1),$ $\text{length}(y(i)) = \text{length}(a)$
31*	<code>musv.m</code>	$y = \text{musv}(sca, vec)$	$sca$ is Mulprec scalar, $vec$ is Mulprec vector $y = sca \cdot vec$
32*	<code>scalp.m</code>	$y = \text{scalp}(vec1, vec2)$	scalar prod. in Euclidean space, $vec1, vec2$ Mulprec column vectors
33	<code>adv.m</code>	$z = \text{adv}(x, y)$	$z = x + y; x, y, z$ Mulprec vectors
34	<code>rnizev.m</code>	$y = \text{rnizev}(x)$	normalizes real Mulprec vector
35	<code>chopv.m</code>	$y = \text{chopv}(x, s)$	chops components of Mulprec vector to length $s$
36	<code>chonizv.m</code>	$y = \text{chonizv}(x, s)$	normalizes and chops a Mulprec vector

### B.2.3 Miscellaneous

No.	m-file	Operation
50*	intro.m	starting routine for Mulprec
51*	rich3.m	Mulprec algorithm for repeated Richardson $h^2$ extrapolation
52*	polygons.m	compute circumference for a sequence of polygons; calls rich3.m

### B.2.4 Using Mulprec

A tar file named `Mulprec.tar` containing the Mulprec m-files can be downloaded from the homepage of the book at [www.siam.org/books/ot103](http://www.siam.org/books/ot103). This tar file also contains some *edited diaries* of a few test experiments (comparisons of computations with different precision), e.g., `pippi2.dia` ( $\pi$  computed by `polygons.m` and `rich3.m`), `muat2est.dia` ( $\pi = 4 \arctan 1$ ), `etest.dia` ( $e$  computed by `expo.m`). It can be easily unpacked so that the separate files become accessible.

To start Mulprec, change the directory to the seat of the Mulprec files. Start MATLAB and run `intro.m`. (*If you forget this*, you are likely to obtain confusing error messages. Ignore them and run `intro.m`!) Then `intro.m` loads the file `const.mat` from the disk. The file `const.mat` contains, e.g., 50 gytes Mulprec approximations to  $\pi$  (called `pilong`) and to  $\ln P$ ,  $P = 10^7$  (called `LP`), and the default values of some other global variables. Now MATLAB is ready for your Mulprec adventures. The first time, you may type “whos” and then type the constants.

It should be kept in mind that the m-files are preliminary versions. Some of them are not thoroughly tested, although most of them were used in successful computations of  $\pi$  to up to 330 decimal places.

## Computer Exercises

- B.1.** Make up and run some simple examples with several choices of the parameter  $s$  such that you can easily check the accuracy of the result. For example:  $1/7$ ,  $\sqrt{0.75}$ ,  $\sin(\pi/3)$ ,  $e$ ,  $4 \arctan 1$ . (Compare also the calculations in the `dia` files.)
- B.2.** A regular  $n$ -sided polygon inscribed in a circle with radius 1 has circumference  $2a_n = 2n \sin(\pi/n)$ . If we put  $h = 1/n$ , then

$$c(h) = c_n = \frac{1}{h} \sin(\pi h) = 2\pi - \frac{\pi^3}{3}h^2 + \frac{\pi^5}{60}h^4 - \dots,$$

and thus  $c_n$  satisfies the assumptions for repeated Richardson extrapolation with  $p_k = 2k$ .

A recursion formula that leads from  $c_n$  to  $c_{2n}$  is given in Example 3.4.13. The script file `polygons.m` uses a similar recursion after the substitutions

$$n = 6 \cdot 2^{m-1}, \quad m = 1 : M, \quad M \leq 36, \quad c_n = p(m+1), \quad q = p/n.$$

The script `polygons.m` then calls the function `rich3.m` that performs Richardson extrapolations until the list of  $M$  polygons is exhausted or the sequence of estimates of the limit  $2\pi$  ceases to be monotonic.

Choose a suitable  $M$ ,  $M \leq 36$ , and call `polygons.m`. Compare with the diary file `pippi2.dia` that contains previous runs of this. Study the elapsed time.

- B.3.** Write a code for the summation of an infinite series by Euler–Maclaurin’s summation formula, assuming that convenient algorithms exist for the integral and for derivatives of arbitrary order. Consider also how to handle generalized cases where a *limit* is asked for, rather than a sum, e.g., Stirling’s asymptotic expansion for  $\log(\Gamma(z))$  or the Euler constant  $\gamma$ .

The numerators and denominators of some Bernoulli numbers  $B_{2n}$ ,  $n = 1 : 17$ , are found in the file `const.dia`, in the vectors `B2nN` and `B2nD`, respectively.  $B_0 = 1$  and  $B_1 = -1/2$  are given separately.

- B.4.** An interesting table of mathematical constants (40 decimal places) is given in Knuth [2, Appendix A]. Compute a few of them to much higher accuracy. For some of them an estimate of the accuracy may be most easily obtained by comparing results obtained using different values of the parameter  $s$ . (Compare also the given diary files.) Some of the constants may require some version of the Euler–Maclaurin formula; see Exercise B.3 above. Incorporate them to your `const.mat` if they are interesting.  $\Gamma(1/3)$  and  $-\zeta'(2)$  (the derivative of Riemann’s  $\zeta$ -function) seem to be relatively advanced tasks.

- B.5.** Write and test a code for the product of a Mulprec matrix by a Mulprec vector. Incorporate it into your Mulprec library.

- B.6.** (a) Implement an operation tentatively called `mullong.m` (`z = mul(x, y, s)`) that can handle a multiplier by partitioning it into 90-gyte pieces and calling `mul.m` once for each piece. Do something about the consequences of this for `expo.m`, if you want to try Exercise B.8.

(b) Implement an operation called `muabs.m` (`z = |x|`) for computing the absolute value of a real or complex Mulprec number.

- B.7.** Write a Mulprec analogue to the MATLAB command `rat` for finding accurate (or exact) rational approximations to floating-point results, in connection with the basic operations of exact rational arithmetic and continued fractions, including `gcd` and `lcm`. (See Knuth [2, Sec. 4.5.2, in particular p. 327]. Mulprec can, of course, not compete with Maple and similar systems for rational arithmetic. Minor tasks of this type may, however, appear in a context where Mulprec is used.

- B.8.** Poisson’s summation formula reads, in the case  $f(t) = e^{-t^2 h^2}$  with the Fourier transform  $\hat{f}(\omega) = (\sqrt{\pi}/h)e^{-\omega^2/(4h^2)}$ ,

$$h \sum_{n=-N}^N e^{-n^2 h^2} = \sqrt{\pi} \sum_{k=-K+1}^{K-1} e^{-\pi^2 k^2 / h^2} + R_{h,N,K, \text{choppings}}.$$

This particular case is also known as the theta transformation formula. To our (limited) knowledge, it has not been previously applied to the high precision computation of good old  $\pi$ .

Suppose that you want to compute  $\sqrt{\pi}$  to an extreme accuracy by letting your desk computer work over a weekend with the use of Mulprec (with a few amendments). For a given (approximate) bound for  $R_{h,N,K, \text{choppings}}$ , determine a good choice of the parameters  $h$ ,  $N$ ,  $K$  and the parameter  $s$  in the various terms. Estimate roughly the relation of computing time to error. Exercise B.6 must have been resolved, at least in principle, before you can solve this.

Before you make a full scale experiment, make sure that neither your computer—nor your office—will be a ruin when you return after the weekend.

*Hints:* Note that the function evaluation can be arranged as a set of recursion formulas with basic arithmetic operations only. We believe that only two or three evaluations of the exponential will be needed in the whole computation.

Leave the door open for the use of variable precision, although this may not reduce the computing time by a terrific amount in this exercise.

Note that  $\pi$  appears in several places in the equation. Think of the computation as an iterative process (although in practice one iteration is perhaps enough).

- B.9.** Implement the qd algorithm and apply it to the classical ill-conditioned problem to transform a power series to a continued fraction or, equivalently, finding the coefficients of the three-term recurrence relation of the orthogonal polynomials for a weight distribution, the moments of which are given (to an enormous number of decimal places).

*MULPREC GIVES YOU  
ALL THE THINGS YOU DO NOT NEED.*

## Bibliography

- [1] Herman H. Goldstine. *A History of Numerical Analysis from the 16th through the 19th Century*. Stud. Hist. Math. Phys. Sci., Vol. 2. Springer-Verlag, New York, 1977. (cited on p. B-4)
- [2] Donald E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, MA, third edition, 1998. (Cited on pp. B-3, B-7.)