MPFR: a multiple-precision floating-point library with correct rounding

(www.mpfr.org)

Paul Zimmermann

Spaces Project, INRIA Lorraine/LORIA, Nancy, France

(with contributions from Sylvie Boldo, David Daney, Alain Delplanque, Mathieu Dutour, Torbjörn Granlund, Guillaume Hanrot, Emmanuel Jeandel, Vincent Lefèvre, Patrick Pélissier, Jean-Luc Rémy, Nathalie Revol, Fabrice Rouillier, Kevin Ryde)

Plan

- some examples
- the IEEE 754 standard
- more examples
- yet another library?
- again more examples
- future plans

How to know the internal basis of your computer?

Proposed by Malcolm (1972), here with Pari 2.1.4:

```
? \p
   realprecision = 28 significant digits
? x = 1.0
%1 = 1.000000000000000000000000000
? while (((x + 1.0) - x) == 1.0, x = 2.0 * x)
? x
%2 = 7.922816251426433759354395033 E28
? y = 1.0
%3 = 1.000000000000000000000000000
? while ((x + y) - x <> y, y = y + 1.0)
? y
%4 = 2.000000000000000000000000000
```

What's the correct sign?

```
foo(n) =
{
default(realprecision, n);
x=77617.0; y=33096.0;
1335*y^6/4+x^2*(11*x^2*y^2-y^6-121*y^4-2)+11*y^8/2+x/(2*y)
}
```

```
? foo(1)
%31 = 2. E27
? foo(9)
%34 = 2.47588007 E27
? foo(10)
%35 = 0.E18
? foo(19)
%44 = 0.E18
? foo(20)
%45 = -134217726.
? foo(28)
%53 = -134217726.
? foo(29)
%54 = -0.827396060
```

Portability

```
leon1(zimmerma): /usr/local/pari-2.0.14/bin/gp
                  GP/PARI CALCULATOR Version 2.0.14 (alpha)
                             Alpha 64-bit version
                (readline enabled, extended help not available)

                          Copyright (C) 1989-1999 by
          C. Batut, K. Belabas, D. Bernardi, H. Cohen and M. Olivier.

jules(zimmerma): /usr/local/pari-2.0.14/bin/gp
                  GP/PARI CALCULATOR Version 2.0.14 (alpha)
                i686 running linux (ix86 kernel) 32-bit version
                (readline enabled, extended help not available)

                          Copyright (C) 1989-1999 by
          C. Batut, K. Belabas, D. Bernardi, H. Cohen and M. Olivier.
```

```
leon1(zimmerma): /usr/local/pari-2.0.14/bin/gp
? \p 20
   realprecision = 38 significant digits (20 digits displayed)
? -392489*Pi-245399*log(2)+1167281*zeta(3)
%9 = -3.0095202332645702251 E-13
```

```
jules(zimmerma): /usr/local/pari-2.0.14/bin/gp
? \p 20
   realprecision = 28 significant digits (20 digits displayed)
? -392489*Pi-245399*log(2)+1167281*zeta(3)
%7 = -3.0095202347814701290 E-13
```

Remark: release 2.1.4 (32-bit version) prints only 19 digits:
`-3.009520234781470129 E-13`.

```
leon1(zimmerma): /usr/local/pari-2.0.14/bin/gp
? \p 18
   realprecision = 19 significant digits (18 digits displayed)
? -392489*Pi-245399*log(2)+1167281*zeta(3)
%1 = -7.95807864051312208 E-13

jules(zimmerma): /usr/local/pari-2.0.14/bin/gp
? \p 18
   realprecision = 19 significant digits (18 digits displayed)
? -392489*Pi-245399*log(2)+1167281*zeta(3)
%1 = -7.95807863958179950 7 E-13
```

The 0.9999 problem

```
? \p 28
? x=10^29
%30 = 100000000000000000000000000000
? x=10.0^29
%37 = 9.999999999999999999999999999 E28
? x=10^29.0
%38 = 9.999999999999999999999999976 E28
? x=10^29+0.0
%43 = 100000000000000000000000000001.
? x=10^29-1.0
%44 = 100000000000000000000000000000.
? x=10.0^29+10
%46 = 1.000000000000000000000000000 E29
```

What is a floating-point (fp) number?

$$x = m \cdot 2^e$$

m is the mantissa: $2^{p-1} \leq m < 2^p$

p is the precision

e is the exponent: $E_{\min} \leq e \leq E_{\max}$

special numbers: $x = \pm 0$, $x = \pm\infty$, $x = \mathrm{NaN}$

The IEEE 754 standard

Discussions started in 1977, standard was approved in 1985.

Cf *An Interview with the Old Man of Floating-Point*, interview from William Kahan (Turing Award 1989) by Charles Severance, `http://cch.loria.fr/documentation/IEEE754/`

Standardize the representation of fp numbers, operations and their semantics, exceptions (invalid operation, division by zero, overflow, underflow, inexact result).

Under revision: `http://www.validlab/com/754R/`

The IEEE 754 standard: Representation

format	total bits	mantissa	exponent	$E_{\min}, E_{\max}$
single precision	32	23+1	8	$-126, 127$
double precision	64	52+1	11	$-1022, 1023$
double extended	80	64	15	$-16382, 16383$
quadruple precision	128	112+1	15	$-16382, 16383$

Implicit leading bit: for "normal" numbers, the leading bit of the mantissa is always 1, and is not stored.

Special values ($\pm 0, \pm\infty, \mathrm{NaN}$, subnormal numbers) correspond to special exponent values.

The IEEE 754 standard: semantics

Let x a fp number: $f(x)$ is not always a fp number.

Four rounding modes: to nearest, towards $-\infty$ (down), towards $+\infty$ (up), towards 0 (truncation). "Even rounding rule" for nearest mode.

Correct rounding: For $\diamond \in \{+, -, \times, \div\}$, and $\sqrt{\ }$, the result of $x \diamond y$ should be the fp number nearest from the exact result for the given rounding mode.

Proposed extension in 754R to fused multiply-add (fma): $x \cdot y + z$.

TwoSum

Example: Fast Two Sum

Proposed by Dekker in 1971, when $|x| \geq |y|$:

$$s \leftarrow x \oplus y$$

$$t \leftarrow s \ominus x$$

$$u \leftarrow y \ominus t$$

$$x + y = s + u$$

Sterbenz

Theorem. If $y/2 \le x \le 2y$, then $x - y$ is exact.

```
? x=Pi/2
%3 = 1.570796326794896619231321691
? y = -zeta(3)
%6 = -1.202056903159594285399738161
? s = x + y
%8 = 0.3687394236353023338315835302
? t = s - x
%9 = -1.202056903159594285399738161
? u = y - t
%12 = 0.E-28
```

And multiple-precision?

```
% maple
    |\^/|     Maple 7 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) 2001 by Waterloo Maple Inc.
 \  MAPLE  /  All rights reserved. Maple is a registered trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> Digits:=3:
> a:=1.0: b:=9e-5:
> Rounding := 0:
> a - b;
                                       1.0
```

And mathematical functions?

```
#include <stdio.h>
#include <math.h>

int
main ()
{
   double d = 1e22;
   printf ("sin(10^22)=%1.20e\n", sin (d));
   return 0;
}

norroy% gcc e.c -lm
```

```
leibniz% ./a.out # PC/Linux
sin(10^22)=4.62613040764601746169e-01

craffe% ./a.out # Sparc/Solaris
sin(10^22)=-8.52200849767188794992e-01

lepois% ./a.out # Alpha/OSF
sin(10^22)=-8.52200849767188790000e-01

athena% ./a.out # MIPS/IRIX
sin(10^22)=0.00000000000000000000e+00

lucrezia% ./a.out # Athlon/Linux
sin(10^22)=4.62613040764601746169e-01
```

And PARI/GP?

```
norroy% gp
                  GP/PARI CALCULATOR Version 2.1.4 (released)
                i686 running linux (ix86 kernel) 32-bit version
                (readline v4.3 enabled, extended help available)

? \p 15
   realprecision = 19 significant digits (15 digits displayed)
? sin(10^22)
  ***   loss of precision in mpsc1.
```

```c
#include "gmp.h"
#include "mpfr.h"

main(int argc, char *argv[])
{
  unsigned int prec = atoi(argv[1]);
  mpfr_t x, y;

  mpfr_init2 (x, prec);
  mpfr_init2 (y, prec);
  mpfr_set_d (x, 1e22, GMP_RNDN);
  mpfr_sin (y, x, GMP_RNDN);
  mpfr_out_str (stdout, 2, 0, y, GMP_RNDN); printf(" (");
  mpfr_out_str (stdout, 10, 0, y, GMP_RNDN); printf(")\n");
}
```

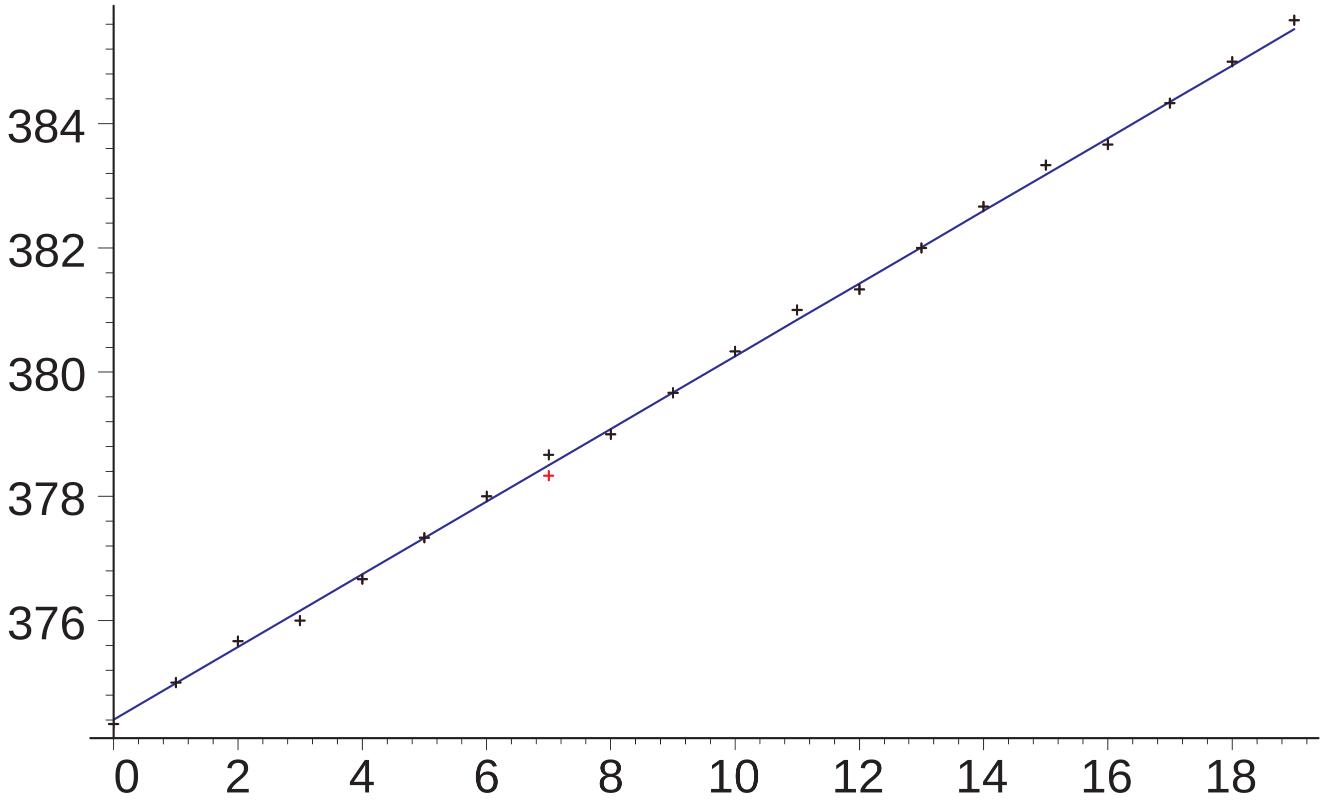
Computing $\sin 10^{22}$

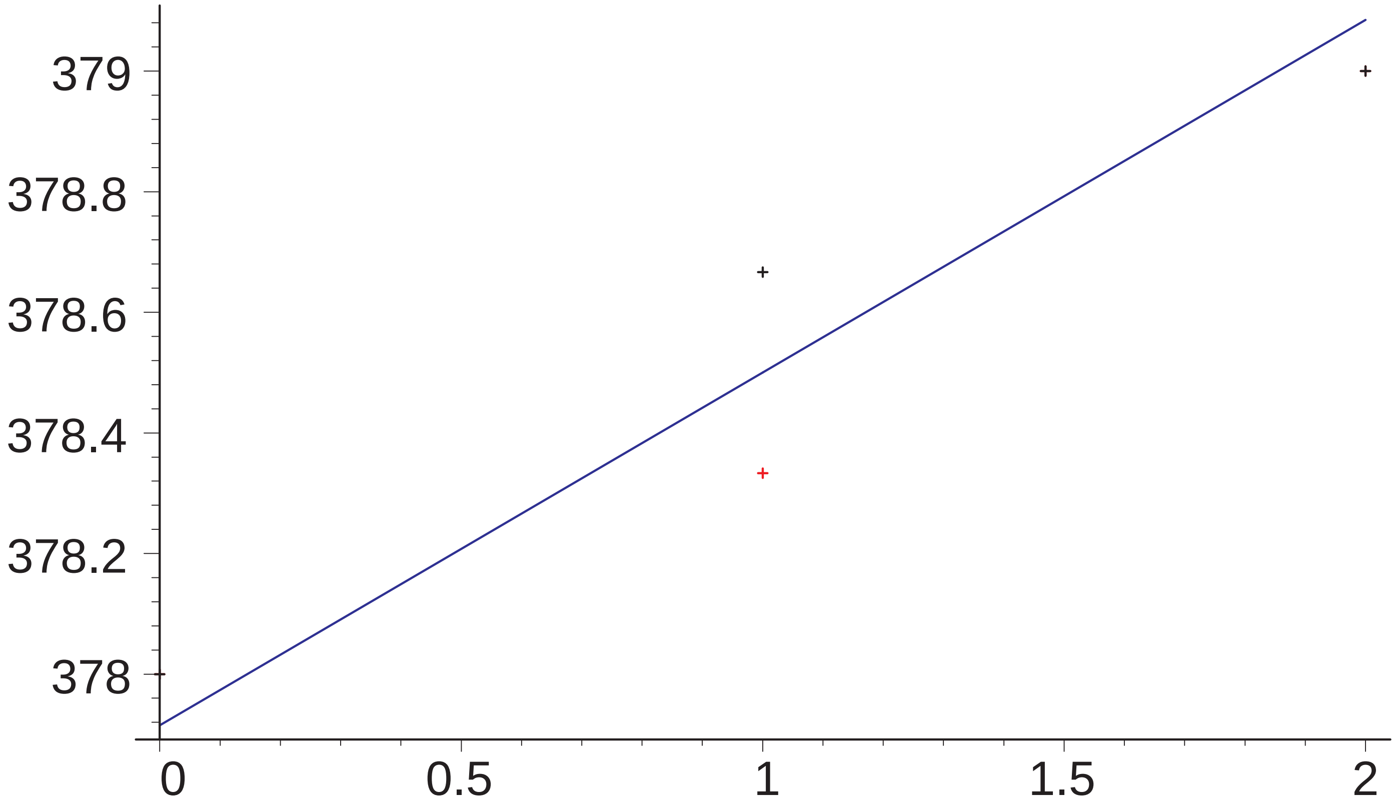
```
[zimmerma@ecrouves tests]$ ./ng 43
-1.111111001010111001000001110100011111101111e-1 (-9.96758488121e-1)

[zimmerma@ecrouves tests]$ ./ng 53
-1.1011010001010011101010110111011010111111001110010111e-1
(-8.52200849767189e-1)

[zimmerma@ecrouves tests]$ ./ng 100
-1.10110100010100111010101101110110101111110011100101110000111111010001\
010011011110010000001110111001010e-1 (-8.522008497671888017727058937 53e-1)
```

Consider $\sin x$ near $x = \frac{1}{2}$.






```
> x := 4503599627371143 / 2^53:
> y := sin(x);
```

$$y := .8636562708040287\textcolor{red}{4998} \cdot 10^{16}$$

The Table Maker's Dilemma

Let x a n-bit fp-number, f a function, $\circ$ a rounding mode

Compute an approximation y of $f(x)$ with precision $m > n$, and error at most one ulp (unit in last place)

How large must be m to ensure that $\circ_n(y) = \circ_n(f(x))$?

Depends on f and n, but also on x...

The Table Maker's Dilemma (continued)

Worst case when $f(x) = \overbrace{b_1.b_2...b_n}^{n}\overbrace{000...000}^{m-n}...$ or $\overbrace{b_1.b_2...b_n}^{n}\overbrace{111...111}^{m-n}...$

Example for $n = 53$:

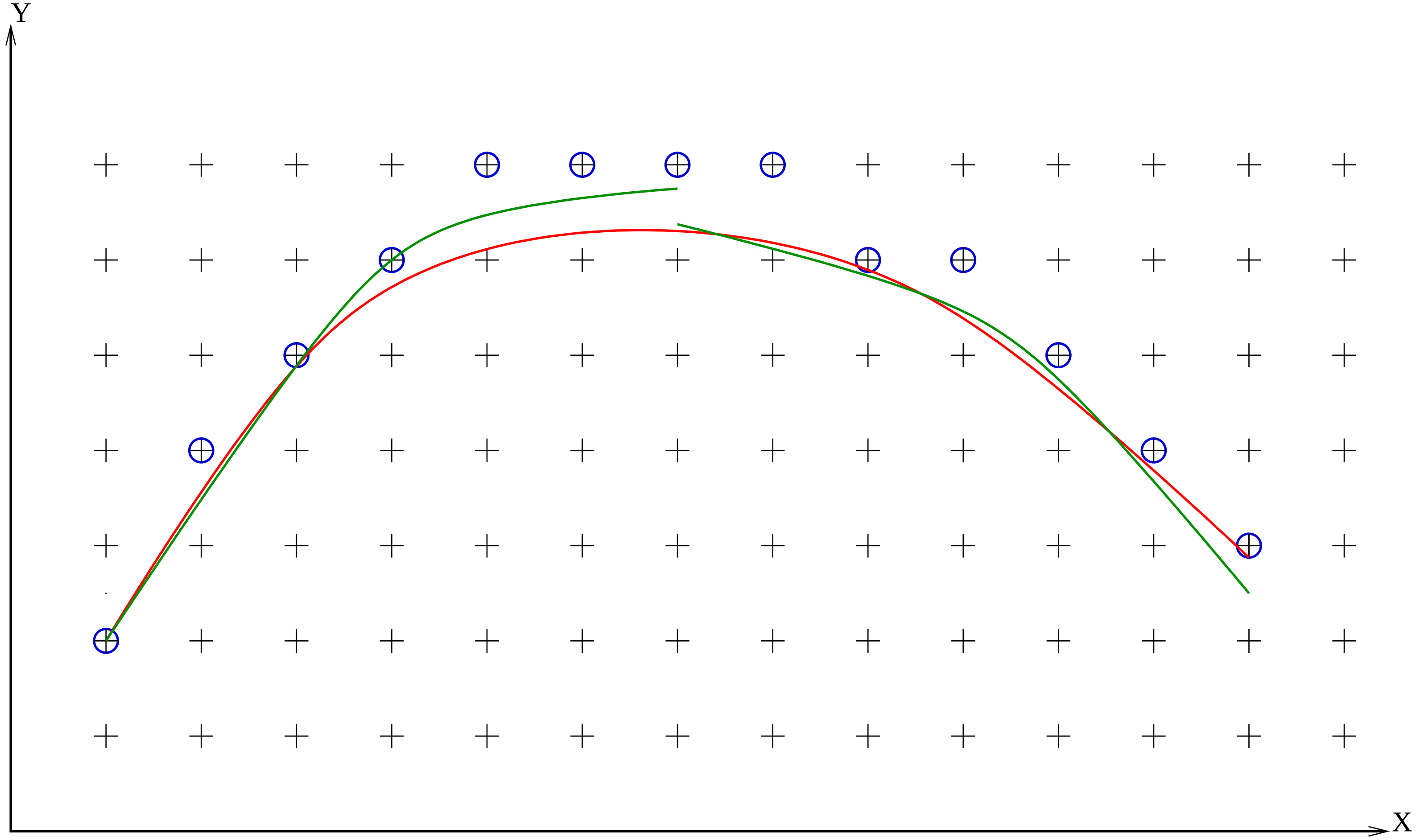
```
> Digits:=105:
> x:=4621447055448553/2^11:
> evalf(arctan(x));
1.57079632679445346798985383429973353983734180715974662500593\
    366229299901038251384413648451282236514126049

> convert(%, binary);
```

1.1001001000011111101101010100010001000010010101001100

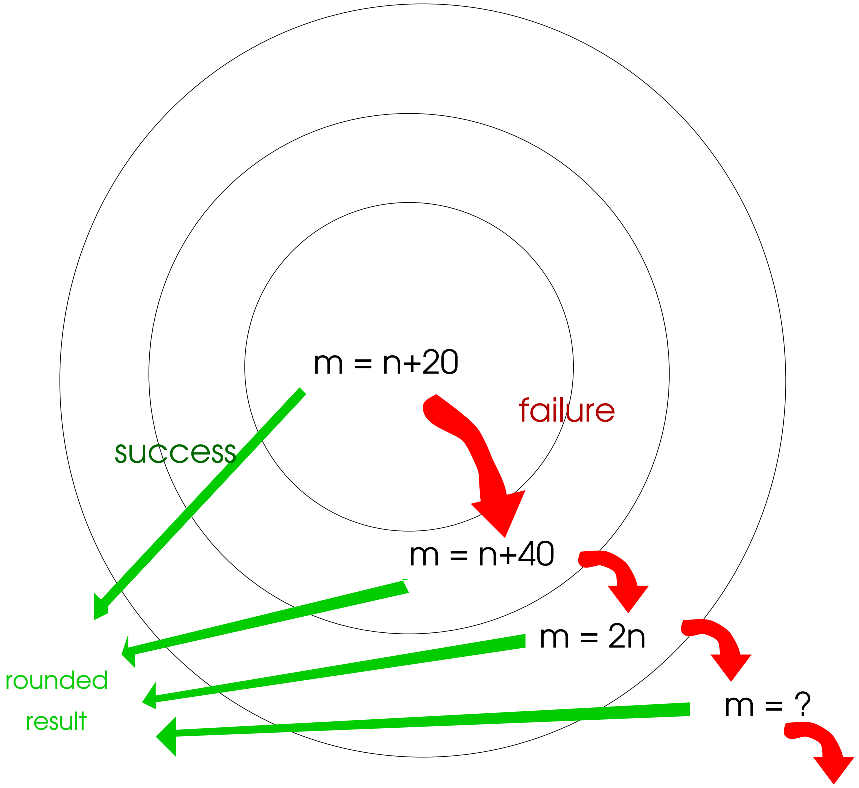
0100000000000000000000000000000000000000000000000000111011

The Table Maker's Dilemma



Arbitrary Precision

Ziv's strategy (*Fast evaluation of elementary mathematical functions with correctly rounded last bit*, ACM TOMS, 1991):



Related work

Elkies (ANTS IV, 2000) to find rational points near a curve, and "counter-examples" to Hall's conjecture

$$|x^2 - y^3| < x^{1/2}$$

Coppersmith (Eurocrypt'96) to find small roots of

$$Q(x, y) \equiv 0 \bmod N$$

MPFR inside

- $+$ and $-$ based on `mpn_add`, `mpn_sub` and `mpn_[lr]shift`: $O(n)$
- $\times$ based on `mpn_mul`: $M(n)$
- $/$ based on `mpn_divrem`: $O(M(n))$
- $\sqrt{\ }$ based on `mpn_sqrtrem`: $O(M(n))$
- agm and log in $O(M(n) \log n)$
- exp has complexity $O(M(n) \log^2 n)$
- $\sin, \cos$ in $O(M(n) \log^2 n)$

MPFR-dev with GMP-4.0, CLN 1.1.3 and with GMP, PARI-2.1.1.

operation	digits	MPFR	CLN	CLN+GMP	PARI
$x \times y$	10^2	0.0009	0.0016	0.0016	0.0010
	10^4	1.03	2.43	1.52	4.71
	10^6	510	810	633	171s
x/y	10^2	0.0024	0.0031	0.0029	0.0024
	10^4	2.19	9.76	4.35	5.57
	10^6	2080	3510	2780	176s
$\sqrt{x}$	10^2	0.0031	0.0030	0.0031	0.0106
	10^4	1.49	3.64	2.60	18.4
	10^6	1570	3340	2590	419s
$\exp(x)$	10^2	0.042	0.115	0.129	0.063
	10^4	135	250	157	1020
	10^6	120s	220s	173s	NA
$\log(x)$	10^2	0.092	0.127	0.155	0.137
	10^4	70.7	243	147	727
	10^6	85s	168s	133s	NA

operation	digits	MPFR	CLN	CLN+GMP	PARI
$\cos(x)$	10^2	0.047	0.123	0.151	0.070
	10^4	197	410	239	1380
	10^6	1226s	326s	256s	NA
$\operatorname{acos}(z)$	10^2	1.398	0.150	0.178	0.291
	10^4	1880	443	269	3980
	10^6	897s	306s	240s	NA
$\log 2$	10^2	0.0004	0.0041	0.0100	0.138
	10^4	0.0009	0.0041	1.54	570
	10^6	2.70	0.004	600	NA
π	10^2	0.0004	0.0003	0.0006	0.0433
	10^4	0.0046	0.0021	0.0023	116
	10^6	2.62	2.17	2.03	NA
γ	10^2	0.855	6e-5	9e-5	1.476
	10^4	4060	3860	1920	7720
	10^6		1467s		NA

Current state and future plans

mpfr-2.0.3 implements input/output ($2 \le \beta \le 36$), conversions from/to integer or double, basic operations ($+, -, \times, /, \sqrt{}$), comparison, $\exp, \log, \mathrm{agm}$, base 2 exponential and logarithm, base-10 logarithm, power, factorial, trigonometric and hyperbolic functions and their inverses, fused multiply-add, $\sqrt{x^2 + y^2}$, $\log(1 + x)$, $\exp(x) - 1$, the constants π and $\log 2$.

Next version will implement:

- missing functions from the ISO C9X standard and other functions (atan2, erfc, lgamma),
- better overflow/underflow handling,
- faster operations for small precision (under 200 bits).

MPtools

- MPFR: arbitrary precision floating-point numbers with correct rounding (`www.mpfr.org`).
- MPFR++ (N. Revol): C++ interface
- MPFI (N. Revol, F. Rouillier): floating-point intervals (web page N. Revol).
- MPC (with Andreas Enge): complex numbers.
- MPcheck (with N. Revol): test properties of mathematical libraries like `libm.a`: correct rounding, monotonicity, symmetry, directed rounding, ...

Rump's "polynomial"

- What is the sign of

$$\frac{1335y^6}{4} + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \frac{11y^8}{2} + \frac{x}{2y}$$

for $x = 77617$ and $y = 33096$?

```
#include <stdlib.h>
#include "gmp.h"
#include "mpfr.h"

void main(int argc, char *argv[]) /* rump precision rnd_mode */
{
  mp_prec_t p;
  mpfr_t x, y, x2, y2, y4, y6, y8, t, tl, tu, s, yu, y2u, y4u, y6u, x2u, y8u, xu;

  p = atoi(argv[1]);
  mpfr_init2(x, p); mpfr_init2(y, p); mpfr_init2(x2, p); mpfr_init2(y2, p);
  mpfr_init2(y4, p); mpfr_init2(y6, p); mpfr_init2(y8, p); mpfr_init2(t, p);
  mpfr_init2(yu, p); mpfr_init2(y2u, p); mpfr_init2(y4u, p); mpfr_init2(y6u,p);
  mpfr_init2(s, p); mpfr_init2(xu, p); mpfr_init2(y8u, p); mpfr_init2(x2u, p);
  mpfr_init2(tu, p); mpfr_init2(tl, p);

  mpfr_set_ui(x, 77617, GMP_RNDD); mpfr_set_ui(xu, 77617, GMP_RNDU);
  mpfr_set_ui(y, 33096, GMP_RNDD); mpfr_set_ui(yu, 33096, GMP_RNDU);
  mpfr_mul(x2, x, x, GMP_RNDD); mpfr_mul(x2u, xu, xu, GMP_RNDU);
  mpfr_mul(y2, y, y, GMP_RNDD); mpfr_mul(y2u, yu, yu, GMP_RNDU);
  mpfr_mul(y4, y2, y2, GMP_RNDD); mpfr_mul(y4u, y2u, y2u, GMP_RNDU);
  mpfr_mul(y6, y4, y2, GMP_RNDD); mpfr_mul(y6u, y4u, y2u, GMP_RNDU);
  mpfr_mul(y8, y4, y4, GMP_RNDD); mpfr_mul(y8u, y4u, y4u, GMP_RNDU);
```

```
/* lower bound of 11x^2y^2 - y^6 - 121y^4 - 2 */
mpfr_mul(t, x2, y2, GMP_RNDD);
mpfr_mul_ui(t, t, 11, GMP_RNDD); /* t <= 11x^2y^2 */
mpfr_sub(t, t, y6u, GMP_RNDD); /* t <= 11x^2y^2 - y^6 */
mpfr_mul_ui(s, y4u, 121, GMP_RNDU); /* s >= 121y^4 */
mpfr_sub(t, t, s, GMP_RNDD); /* t <= 11x^2y^2 - y^6 - 121y^4 */
mpfr_set_ui(s, 2, GMP_RNDD);
mpfr_sub(tl, t, s, GMP_RNDD); /* tl <= 11x^2y^2 - y^6 - 121y^4 - 2 */

/* upper bound of 11x^2y^2 - y^6 - 121y^4 - 2 */
mpfr_mul(t, x2u, y2u, GMP_RNDU);
mpfr_mul_ui(t, t, 11, GMP_RNDU); /* t >= 11x^2y^2 */
mpfr_sub(t, t, y6, GMP_RNDU); /* t >= 11x^2y^2 - y^6 */
mpfr_mul_ui(s, y4, 121, GMP_RNDD); /* s >= 121y^4 */
mpfr_sub(t, t, s, GMP_RNDU); /* t >= 11x^2y^2 - y^6 - 121y^4 */
mpfr_set_ui(s, 2, GMP_RNDU);
mpfr_sub(tu, t, s, GMP_RNDU); /* tu >= 11x^2y^2 - y^6 - 121y^4 - 2 */

/* compute lower bound */
mpfr_mul(t, x2, tl, GMP_RNDD); /* t <= x^2*(11x^2y^2 - y^6 - 121y^4 - 2) */
mpfr_mul_ui(s, y6, 1335, GMP_RNDD);
mpfr_div_2exp(s, s, 2, GMP_RNDD); /* s <= 1335y^6/4*/
mpfr_add(s, s, t, GMP_RNDD); /* s <= 1335y^6/4 + x^2*(...) */
```

```
mpfr_mul_ui(t, y8, 11, GMP_RNDD);
mpfr_div_2exp(t, t, 1, GMP_RNDD); /* t <= 11y^8/2 */
mpfr_add(s, s, t, GMP_RNDD); /* s <= 1335y^6/4 + x^2*(...) + 11y^8/2 */
mpfr_div(t, x, yu, GMP_RNDD); mpfr_div_2exp(t, t, 1, GMP_RNDD); /* t<= x/2y*/
mpfr_add(s, s, t, GMP_RNDD);
mpfr_out_str(stdout, 10, 0, s, GMP_RNDD);
printf(" <= R <= ");

/* compute upper bound */
mpfr_mul(t, x2u, tu, GMP_RNDU); /* t <= x^2*(11x^2y^2 - y^6 - 121y^4 - 2) */
mpfr_mul_ui(s, y6u, 1335, GMP_RNDU);
mpfr_div_2exp(s, s, 2, GMP_RNDU); /* s <= 1335y^6/4*/
mpfr_add(s, s, t, GMP_RNDU); /* s <= 1335y^6/4 + x^2*(...) */
mpfr_mul_ui(t, y8u, 11, GMP_RNDU);
mpfr_div_2exp(t, t, 1, GMP_RNDU); /* t <= 11y^8/2 */
mpfr_add(s, s, t, GMP_RNDU); /* s <= 1335y^6/4 + x^2*(...) + 11y^8/2 */
mpfr_div(t, xu, y, GMP_RNDU); mpfr_div_2exp(t, t, 1, GMP_RNDU); /* t<= x/y */
mpfr_add(s, s, t, GMP_RNDU);
mpfr_out_str(stdout, 10, 0, s, GMP_RNDU);
putchar('\n');

mpfr_clear(x); mpfr_clear(y); mpfr_clear(x2); mpfr_clear(y2); mpfr_clear(yu);
mpfr_clear(y4); mpfr_clear(y6); mpfr_clear(y8); mpfr_clear(t);
```

```
  mpfr_clear(y2u); mpfr_clear(y4u); mpfr_clear(y6u); mpfr_clear(s);
  mpfr_clear(xu); mpfr_clear(y8u); mpfr_clear(x2u); mpfr_clear(tu);
  mpfr_clear(tl);
}
```

Sign of Rump's polynomial

```
% ./rump 1
-2.e39 <= R <= 6.e39

% ./rump 121
-2.82739605994682136814116509547981630 <= R <=
1.17260394005317863185883490452018371

% ./rump 122
-8.27396059946821368141165095479816293e-1 <= R <=
-8.27396059946821368141165095479816291e-1
```

It's now open to you

at `www.mpfr.org/cvs.html`