Well-defined semantics for floating-point computations: the MPFR library

Guillaume Hanrot, Fabrice Rouillier, Paul Zimmermann

INRIA Lorraine and LORIA, Nancy, France

(with contributions from Jean-Luc Rémy, Sylvie Boldo, Emmanuel Jeandel)

http://www.loria.fr/projets/mpfr/

```
    |\^/|     Maple 6 (IBM INTEL LINUX22)
._|\|   |/|_. Copyright (c) 2000 by Waterloo ...
 \  MAPLE  /  All rights reserved. Maple is a ...
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.

> p:=1335*y^6/4+x^2*(11*x^2*y^2-y^6-121*y^4-2)
  +11*y^8/2+x/2/y:

> Digits:=1: subs(x=77617.,y=33096.,p);
                        1.

> Digits:=2: subs(x=77617.,y=33096.,p);
                          36
                     -.2 10

> Digits:=4: subs(x=77617.,y=33096.,p);
                         34
                     .3 10

> Digits:=5: subs(x=77617.,y=33096.,p);
                          33
                     -.1 10

> Digits:=8: subs(x=77617.,y=33096.,p);
                         30
                     .1 10
```

```
Mathematica 4.0 for Linux
Copyright 1988-1999 Wolfram Research, Inc.
 -- Motif graphics initialized --

In[1]:= p=1335*y^6/4+x^2*(11*x^2*y^2-y^6
           -121*y^4-2)+11*y^8/2+x/2/y;

In[2]:= d=1; p /. {x->N[77617,d],y->N[33096,d]}

                  21
Out[2]= 1.18059 10

In[4]:= d=17; p /. {x->N[77617,d],y->N[33096,d]}

             21
Out[4]= 0. 10

In[25]:= d=38; p /. {x->N[77617,d],y->N[33096,d]}

Out[25]= 0.

In[26]:= d=39; p /. {x->N[77617,d],y->N[33096,d]}

Out[26]= -0.8
```

Exact rounding - IEEE 754

Floating point number

(fp) $= \pm m \cdot 2^e$, m 53 bit unsigned integer, e 11 bit integer.

x and y fp $\not\Rightarrow$ $(x+y)$ fp, xy fp, $\ldots$.

The IEEE 754 norm aims at giving a precise semantics to floating point operations $(+, -, \times, /, \sqrt{})$.

Exact rounding concept:

the result of $f(x,y)$ should be equal to the result of first computing *exactly* $f(x,y)$ and then rounding m.

Example :

$\sqrt{4761827905759444} = 4630914205854029.999911 \cdot 2^{-26}$ has to yield $4630914205854029 \cdot 2^{-26}$ when rounded down.

MPFR intends to provide operations with similar semantics for multiprecision numbers (each variable having its own precision).

Installation procedure

GNU autoconf supported in the forthcoming version (0.5). In versions ≤ 0.4 a basic homemade Configure script is supplied, suitable for classical architectures.

```
latour ~/mpfr $ ./configure --with-gmp=...
...
creating Makefile
creating tests/Makefile
latour ~/mpfr $ make
...
```

Currently tested architectures: i686/Linux 2.2.x, Alpha ev6/OSF1, SGI R10000/IRIX 6.5, Sun UltraSparc / Solaris 2.6, HP 9000/HP-UX A.09.05, ix86/Windows (Cygnus compiler).

Wanted : feedback on other architectures.

Licence

MPFR is distributed under the GNU Library General Public License.

MPFR and GMP-3.x

MPFR is based on T. Granlund's excellent GMP library.

GMP is a multiprecision package which

- is highly portable,
- is probably the fastest available on a wide range of architectures,
- provides easy access to low level functions.

We hope that MPFR has inherited or shall inherit the first two of these characteristics.

Experimental results : in "average" situations, around 10% loss of efficiency.

Features

MPFR currently implements

- basic arithmetic : +, -, $\times$, / ;
- $\sqrt{}$, agm ;
- log, exp, sin, cos ;
- generic code for a whole subclass of hypergeometric functions.

Projects:

- support for `NaN` and `Inf`.
- reciprocal circular functions ;
- (put your dearest wishes here).

Hypergeometric functions

Generic code (based on E. Karatsuba's binary splitting method) for functions $\sum a_n z^n$ when $a_{n+1}/a_n = P(n)/Q(n)$, P and Q with rational roots, degrees ≤ 2. Allows to obtain $\exp, \log, \cos, \ldots$

```
#define A              /* P has degree >= 1 */
#define A1 1
#define A2 1           /* (n + A1/A2) divides P */
#undef  B              /* P has degree 1, = (n + 1) */
#define C
#define C1 2
#define C2 1           /* (n + C1/C2) divides Q */
#define NO_FACTORIAL /* No (n+1) in Q, Q = (n+2) */
#define GENERIC mpfr_aux_log2
#include "../generic.c"
...
mpz_init_set_ui(z, 1);
mpfr_aux_log2(x, z, 1, 10);
```

yields the sum of 2^{10} terms of $-\log(1-x)/x$ evaluated at $1/2^1$: 1.38629436111989

An example of MPFR code

```
void cfpi(unsigned long prec) {
  mpfr_t pi, pi2, a, a2, temp;

  mpfr_init2(pi, prec); mpfr_init2(pi2, prec);
  mpfr_init2(a, prec); mpfr_init2(a2, prec);
  mpfr_init2(temp, prec);

  mpfr_const_pi(pi, prec, GMP_RNDD);
  mpfr_set(pi2, pi, GMP_RNDU);
  mpfr_add_one_ulp(pi2);

  mpfr_floor(a, pi); mpfr_floor(a2, pi2);
  while(mpfr_cmp(a, a2) == 0) {
    mpfr_sub(pi, pi, a, GMP_RNDD);
    mpfr_sub(pi2, pi2, a2, GMP_RNDU);
    mpfr_ui_div(temp, 1, pi, GMP_RNDU);
    mpfr_ui_div(pi, 1, pi2, GMP_RNDD);
    mpfr_set(pi2, temp, GMP_RNDU);

    printf("%d\n", (int)mpfr_get_d(a));
    mpfr_floor(a, pi); mpfr_floor(a2, pi2);
  }
}
```

100 random tests in double precision

`mpfr_log` differs from `libm.a` for

```
 a=2.79525179610122883102e-76, rnd_mode=GMP_RNDD
libm.a   : -1.73968544876947561306e+02
mpfr_log : -1.73968544876947589728e+02 (-1 ulp)
...
```

`mpfr_log` differs from `libm.a` for

```
 a=4.60768080306399470321e-41, rnd_mode=GMP_RNDU
libm.a   : -9.28782641619299766944e+01
mpfr_log : -9.28782641619299624836e+01 (1 ulp)
max error : 1   mean error : 0.380000  (in ulps)
```

`mpfr_exp` differs from `libm.a` for

```
 x=-6.33049190641098107335e+02, rnd=GMP_RNDD
libm.a   : 1.17551921166667329222e-275,
mpfr_exp : 1.17551921166672411793e-275 (317 ulp)
...
```

`mpfr_exp` differs from `libm.a` for

```
 x=4.67812749656642154150e+02, rnd=GMP_RNDU
libm.a   : 1.47399408202541049400e+203,
mpfr_exp : 1.47399408202530624206e+203 (599 ulp)
max error=599 mean error=1.54e+02
```

How to update your MPF code to MPFR ?

Just add the following after `#include "gmp.h"`:

```
#include "mpfr.h"
#include "mpf2mpfr.h"
```

The rounding mode used is the default one (to nearest initially) and can be changed as follows:

```
mpfr_set_default_rounding_mode(GMP_RNDZ);
```

With MPSolve (`www.dm.unipi.it/pages/bini`):

```
$ time ./unisolve < DATA/curtz100.pol
(-0.287735795654237869 2e0, 0.424038293143887484 1e0)
...
(0.135327803437946195 5e1, 0.296667230853870675 2e-1)
5.360u 0.010s 0:05.38 99.8%     0+0k 0+0io 151pf+0w

$ time ./unisolve.mpfr < DATA/curtz100.pol
Using MPFR with rnd=GMP_RNDZ
(-2.877357956022024154e-1, -4.240382928401231765e-1)
...
(1.353278033435344696, 2.966672307229600846e-2)
9.180u 0.020s 0:09.20 100.0%    0+0k 0+0io 154pf+0w
```

Related work

Arithmetic Explorer (Univ. Antwerp, Belgium)

`win-www.uia.ac.be/u/cuyt/arexp.html`

and Arithmos (commercial version)

The RealRam implementation from N. Müller

`www.informatik.uni-trier.de/~mueller/`

The Table Maker Dilemma

`www.ens-lyon.fr/~jmuller/Intro-to-TMD.htm`

High Radix Online Arithmetic by Tom Lynch

`devil.ece.utexas.edu/~lynch/online/online.html`

References

[1] Sofroniou, M., and Spaletta, G. Precise numerical computation, Nov. 1999. Preprint. 18 pages.

[2] Zimmermann, P. Karatsuba square root. Research Report 3805, INRIA, Nov. 1999.