MPFR : une bibliothèque de calcul flottant en précision arbitraire avec arrondi correct

(www.mpfr.org)

Paul Zimmermann

Projet Spaces, INRIA Lorraine/LORIA

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

La norme IEEE 754

Cf exposé de David (Defour) :

- format flottant (signe, mantisse, exposant)
- paramètres (précision, $E_{\min}$, $E_{\max}$)
- arrondi correct
- valeurs spéciales (NaN, $\pm\infty$, ± 0)
- exceptions (underflow, overflow, inexact, invalid, division by zero)
- nombres dénormalisés

Objectifs de MPFR

- extension de IEEE 754 à la précision arbitraire (sauf dénormalisés)
- portable
- aussi efficace que les concurrents (CLN, PARI/GP, NTL)

État des lieux

Scilab utilise l'arithmétique flottante double précision de la machine et les fonctions mathématiques de `libm.a`.

Sous réserve du respect de IEEE 754 (processeur, système d'exploitation, compilateur), les quatre opérations $(+, -, \times, \div)$ et la racine carrée sont portables.

Attention : problème de précision étendue sur x86 (on ignore ce problème par la suite).

Et les fonctions mathématiques ?

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

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

% gcc e.c -lm
```

```
leibniz% ./a.out # x86/Linux
sin(10^22)=4.626130407646017461 69e-01

craffe% ./a.out # Sparc/Solaris
sin(10^22)=-8.522008497671887949 92e-01

lepois% ./a.out # Alpha/OSF
sin(10^22)=-8.522008497671887900 00e-01

athena% ./a.out # MIPS/IRIX
sin(10^22)=0.000000000000000000 00e+00

lucrezia% ./a.out # Athlon/Linux
sin(10^22)=4.626130407646017461 69e-01
```

Entrées-sorties

Pour les entrées-sorties (conversions binaire-décimal), en double précision, la (révision de la) norme IEEE 754 impose l'arrondi correct pour les nombres décimaux de la forme $\pm M \cdot 10^{\pm N}$ pour $M \leq 10^{17} - 1$ et $N \leq 999$.

Pas encore implanté dans la `libc` actuelle, notamment pour les arrondis dirigés :

```
#include <stdio.h>
#include <stdlib.h>
#include <fenv.h>

int
main ()
{
  double x;
```

```
  fesetround (FE_TOWARDZERO);
  x = strtod ("0.1", NULL);
  printf ("%1.20e\n", x);
  fesetround (FE_UPWARD);
  x = strtod ("0.1", NULL);
  printf ("%1.20e\n", x);
  fesetround (FE_TOWARDZERO);
  printf ("%1.6e\n", x);
  fesetround (FE_UPWARD);
  printf ("%1.6e\n", x);
}

% gcc e.c -lm; ./a.out
1.00000000000000005551e-01
1.00000000000000005551e-01
1.000000e-01
1.000000e-01
```

Écriture pour relecture

Problème : comment stocker un nombre flottant dans un fichier, et obtenir la même valeur à la relecture ?

Solution 1 : le stocker en binaire, octal, hexadécimal.

Solution 2 : si x est converti en y, qui est relu en x', avec arrondi correct au plus proche, il suffit que $\mathrm{ulp}(y) < \mathrm{ulp}(x)$. En effet, $|x - y| \leq \frac{1}{2}\mathrm{ulp}(y) < \frac{1}{2}\mathrm{ulp}(x)$, donc $x' = x$. Il suffit d'avoir

$$P \geq 1 + p\frac{\log 2}{\log 10}.$$

pour $p = 53$, cela donne $P \geq 17$.

Donc si on stocke un flottant double précision avec (au moins) 17 chiffres décimaux, on est certain de relire la même valeur (si arrondi correct).

Le « plus produit »

• portabilité des fonctions mathématiques (idem CRLIBM, mais en précision arbitraire)

• étude de la sensibilité d'algorithmes à la précision (estimation de l'erreur d'arrondi, plus rigoureux encore avec MPFI).

Exemple

Gauss naïf sur matrice de Hilbert ($m_{ij} = \frac{1}{i+j-1}$).

Matrice de dimension n, précision p.

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

int
main (int argc, char *argv[]) {
  int n = atoi (argv[1]), p = atoi (argv[2]), i, j, k;
  mpfr_t **m, t, u, d;

  mpfr_set_default_prec (p);

  m = malloc (n * sizeof (mpfr_t*));
```

```
for (i = 0; i < n; i++) {
    m[i] = malloc (n * sizeof (mpfr_t));
    for (j = 0; j < n; j++) {
        mpfr_init_set_ui (m[i][j], i + j + 1, GMP_RNDN);
        mpfr_ui_div (m[i][j], 1, m[i][j], GMP_RNDN);
      }
  }
mpfr_init (t);
mpfr_init (u);
mpfr_init_set_ui (d, 1, GMP_RNDN);

for (i = 0; i < n; i++) {
    mpfr_mul (d, d, m[i][i], GMP_RNDN);
    for (k = i + 1; k < n; k++) {
        mpfr_div (t, m[k][i], m[i][i], GMP_RNDN);
        for (j = i + 1; j < n; j++) {
            mpfr_mul (u, t, m[i][j], GMP_RNDN);
```

```
            mpfr_sub (m[k][j], m[k][j], u, GMP_RNDN);
          }
      }
  }

mpfr_out_str (stdout, 10, 3, d, GMP_RNDN); printf ("\n");

for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++)
      mpfr_clear (m[i][j]);
    free (m[i]);
  }
free (m);
mpfr_clear (t);
mpfr_clear (u);

return 0;
```

```
}
```

```
% ./gauss 20 53
4.26e-196
% ./gauss 20 100
4.21e-226
% ./gauss 20 200
4.21e-226
```

n	p	det	temps
20	100	4.21e-226	0.001
50	100	-7.44e-1054	0.018
50	200	-6.17e-1435	0.027
50	300	1.39e-1466	0.040
100	600	3.37e-5942	0.754
200	1000	2.94e-23924	12.7

MPFR inside

Un flottant MPFR est une structure C, actuellement :

- précision (en bits) [`unsigned long`]
- signe [`int`]
- exposant [`long`]
- un pointeur sur la mantisse [`mp_limb_t*`]

Le type `mpfr_t` est en fait un pointeur sur un tableau,

```
typedef __mpfr_struct mpfr_t[1];
```

de sorte que la déclaration `mpfr_t x` alloue directement la structure sur la pile.

MPFR inside (suite)

La mantisse est allouée par :

```
mpfr_init2 (x, prec);
```

et libérée par :

```
mpfr_clear (x);
```

L'espace mémoire utilisé par un flottant MPFR ne varie pas au cours d'un calcul (sauf usage de `mpfr_set_prec`).

La gestion de la mémoire est faite via les fonctions d'allocation de GMP (par défaut malloc/realloc/free).

Interface avec MPFR

Certains logiciels (comme Pari/GP) ont leur propre gestionnaire de mémoire. Propositions d'interface :

- interface 1 (minimale)
- interface 2 (maximale)

Interface minimale

```
/* Return the needed size for the mantissa (in bytes) */
size_t mpfr_low_level_get_size (mp_prec_t prec);

/* Init a mpfr_t x using m as the mantissa and p as the precision.
   m must have at least mpfr_low_level_get_size(p) bytes.
   It must be suitably aligned for any kind of variable (e.g. malloc).
   x is set to NAN.
   It uses m for storing the mantissa in further computations on x.
   m is "cleared" (any previous computation is lost). */
void mpfr_low_level_init (mpfr_t x, mp_prec_t p, void *m);

/* Clear a mpfr_t initialized by mpfr_low_level_init.
   Does nothing in the current implementation but it may change. */
void mpfr_low_level_clear (mpfr_t x);
```

Exemple d'utilisation

```
mpfr_t x;
void *m;
double y;

m = (void*) scilab_malloc (mpfr_low_level_get_size (prec));
mpfr_low_level_init (x, p, m);

mpfr_ui_pow_ui (x, 10, 22, GMP_RNDN);
mpfr_sin (x, x, GMP_RNDN);
y = mpfr_get_d (x, GMP_RNDN);

mpfr_low_level_clear (x);
scilab_free (m);
```

Interface (suite)

Les fonctions MPFR peuvent allouer temporairement (via les fonctions de GMP).

Exception : les constantes π, $\log 2$, γ sont « cachées » (appeler `mpfr_free_cache` pour libérer).

Interface maximale

```
/* Return the needed size for the mantissa */
size_t mpfr_mantissa_get_size (mp_prec_t p);

/* Init a mantissa for mpfr.
 * 'm' must have at least mpfr_mantissa_get_size(p) bytes.
 * It must be suitably aligned for any kind of variable. */
void mpfr_mantissa_init (void *m, mp_prec_t p);

/* Clear a mantissa for MPFR */
void mpfr_mantissa_clear (void *m, mp_prec_t p);

/* Perform a dummy initialization of a mpfr_t and set it to NAN
 * It uses 'm' directly for further computing
 * involving 'x'. It won't alloc anything. */
```

```
void mpfr_mantissa_init_fr1 (mpfr_t x, mp_prec_t p, void *m);

/* Perform a dummy initialization of a mpfr_t and set it to a value.
 * if ‘ABS(s) == 0’, ‘x’ is set to NAN.
 * if ‘ABS(s) == 1’, ‘x’ is set to INF of sign sign(s)
 * if ‘ABS(s) == 2’, ‘x’ is set to ZERO of sign sign(s)
 * if ‘ABS(s) == 3’, ‘x’ is set to a regular number: x=sign(s)*m*2^exp
 * In all cases, it uses ‘m’ directly for further computing
 * involving ‘x’. It won’t alloc anything.  */
void mpfr_mantissa_init_fr2 (mpfr_t x, int s, mp_exp_t exp,
     mp_prec_t p, void *m);

/* Return the mantissa  */
void *mpfr_mantissa_get (mpfr_t x);

/* Inform MPFR that the mantissa has moved */
void mpfr_mantissa_after_move (mpfr_t x, void *new_position);
```

Toolbox Matlab

Une toolbox MPFR existe déjà pour Matlab (auteur Ben Barrowes) :

`https://sourceforge.net/projects/mptoolbox/`

Investigations de Serge :

- toutes les opérations passent par des chaînes de caractères
- peu efficace, et perte possible de précision (si base différente)
- boucles interprétées en Matlab
- arrondi fixé (au plus proche)

Conclusion

- intégration *a priori* facile, via l'interface bas niveau
- création d'un type `mpfr` en Scilab ?
- précision globale et/ou locale à chaque variable ?
- arrondi global et/ou spécifique à chaque opération ?
- quid de NaN, Inf ?