Bibliothèque GNU MPFR
Arithmétique flottante en précision arbitraire

Vincent LEFÈVRE

Pascaline, Inria Lyon / LIP, ENS-Lyon

Campus du Libre 2026, 2026-11-21

GNU MPFR en quelques mots

- GNU MPFR est une bibliothèque de calcul en virgule flottante efficace en précision arbitraire avec une sémantique bien définie.

  (Virgule flottante : méthode d'approximation des nombres réels.)

- Généralisation à la précision arbitraire des idées de la norme IEEE 754 (qui couvre les formats implantés dans les processeurs, e.g. double précision).

  En particulier :

  - **arrondi correct** (au plus près ou dans une direction donnée) ;
  - valeurs spéciales : *Not a Number* (NaN), infinis, zéros signés.

- Précision : de 1 bit jusqu'aux limites de la machine.

- Environ 100 fonctions mathématiques, en plus de fonctions utilitaires.

GNU MPFR en quelques mots [2]

- Bibliothèque développée de 1999 à 2006 au LORIA, Inria Nancy.
  Depuis 2006, projet commun entre 2 équipes-projets :
  - à Nancy : Caramba, développement par Paul Zimmermann (1999-) ;
  - à Lyon : Pascaline, développement par Vincent Lefèvre (fin 2000-).

  Autres principaux contributeurs : Guillaume Hanrot (1999-2003), Patrick Pélissier (2003-2005), Philippe Théveny (2007-2009), Matteo Nicoli (2025-).

- Écrite en C (ISO + extensions optionnelles) + appels à GMP pour l'efficacité (couches mpn/mpz : multiprécision sur les entiers).

- Licence : LGPL (version 3 ou plus). Projet GNU depuis 2009.

- Web : `https://www.mpfr.org/`

Historique

1998–2000	ARC INRIA *Fiable*.
Novembre 1998	Texte fondateur (Guillaume Hanrot, Jean-Michel Muller, Joris van der Hoeven, Paul Zimmermann).
Début 1999	Premières lignes de code.
Février 2000	Première version publique (licence LGPL v2+).
Juin 2000	Cession du copyright à la Free Software Foundation.
2004	GNU Fortran utilise MPFR (pour l'arrondi correct).
Mars 2008	GCC utilise MPFR dans son middle-end (pour l'arrondi correct).
Janvier 2009	Sortie de GNU MPFR 2.4.0 : maintenant un paquet GNU.
Juin 2010	Sortie de GNU MPFR 3.0.0 : première version en LGPL v3+.
Mars 2025	Sortie de GNU MPFR 4.2.2 (dernière version).
Décembre 2025	Prix science ouverte du logiciel libre de recherche 2025 attribué à GNU MPFR dans la catégorie « Scientifique et technique ».

Fonctions mathématiques fournies

- Opérations arithmétiques : $+$, $-$, $\times$, $\div$, FMA, somme de n termes.
- Factorielle, puissance, racine n-ième.
- Exponentielle et logarithme en base e, 2 et 10 (idem pour expm1 et log1p).
- Diverses fonctions trigonométriques et hyperboliques et leurs inverses.
- Fonctions spéciales : fonctions gamma et digamma, exponentielle intégrale, dilogarithme, fonction bêta, fonction zêta de Riemann, fonctions d'erreur erf et erfc, fonctions de Bessel (cylindriques) de première et de seconde espèce, moyenne arithmético-géométrique, fonction d'Airy (limitée).
- Fonctions d'arrondi à un entier et de reste.
- Génération de nombres pseudo-aléatoires (distr. uniforme ou gaussienne).
- Pour MPFR 4.3.0 : fonction compound $(1+x)^n$, fonction trigamma, polynômes de Legendre et d'Hermite.

Certaines fonctions ont des variantes plus simples. Exemple : racine carrée.

Pourquoi MPFR ?

En général, les calculs exacts sur les nombres réels ne sont pas possibles : ce serait trop lent, ou même indécidable.

Un système à virgule flottante en précision arbitraire permet une grande plage de valeurs en grande précision.

Critères :

- performance (en temps et mémoire) ;
- précision (*accuracy*), et l'exactitude, le cas échéant ;
- portabilité et reproductibilité (en particulier, les machines 32 bits et 64 bits doivent donner les mêmes résultats, ce qui n'est pas le cas de la couche mpf de GMP, même pour la génération de nombres pseudo-aléatoires).

Compromis entre les différents critères.

Un autre point est apparu : l'émulation d'arithmétiques à précision fixe, comme celle spécifiée par la norme IEEE 754.

Exemple : $\sin(10^{22})$ il y a plus de 20 ans

Environnement	Valeur calculée de $\sin 10^{22}$
Résultat exact	$-\mathbf{0.852200849767188801772}7\ldots$
MPFR sur 53 bits	$-0.8522008497671887 9$
Glibc 2.3.6 / x86	0.4626130407646017 5
Glibc 2.3.6 / x86_64	$-0.8522008497671887 9$
Mac OS X 10.4.11 / PowerPC	$-0.8522008497790920 5$
Maple 10 (Digits = 17)	$-0.8522008497671888 0$
Mathematica 5.0 (x86 ?)	0.462613
MuPAD 3.2.0	-0.9873536182
HP 700	0.0
HP 375, 425t (4.3 BSD)	$-0.65365288\ldots$
Solaris/SPARC	$-0.852200849\ldots$
IBM 3090/600S-VF AIX 370	0.0
PC : Borland TurboC 2.0	4.67734e−240
Sharp EL5806	-0.090748172

Note : $10^{22} = 5^{22} \times 2^{22}$, et 5^{22} est représentable sur 53 bits.

Programme MPFR pour calculer $\sin(10^{22})$

```
#include <stdio.h>   /* pour mpfr_printf, avant #include <mpfr.h> */
#include <assert.h>
#include <mpfr.h>

int main (void)
{
  mpfr_t x;  /* mpfr_t: tableau à 1 élément */
  int inex;

  mpfr_init2 (x, 53);  /* x: précision de 53 bits */
  inex = mpfr_set_ui (x, 10, MPFR_RNDN);     assert (inex == 0);
  inex = mpfr_pow_ui (x, x, 22, MPFR_RNDN);  assert (inex == 0);
  mpfr_sin (x, x, MPFR_RNDN);
  mpfr_printf ("sin(10^22) = %.17Rg\n", x);
  mpfr_clear (x);
  return 0;
}
```

Donner `-lmpfr` à la compilation (`-lmpfr -lgmp` avec certains linkers).

Représentation et arrondi

Règles de la norme IEEE 754, avec quelques différences :

- Base 2, précision $p \geqslant 1$. Chaque nombre MPFR (objet de type `mpfr_t`) a sa propre précision.
- Format des nombres normaux : $\pm \mathbf{0.} \underbrace{\mathbf{1} b_2 b_3 \ldots b_p}_{p \text{ bits}} \cdot \mathbf{2}^e$ avec $E_{\min} \leqslant e \leqslant E_{\max}$

  où $b_i \in \{0, 1\}$ pour $2 \leqslant i \leqslant p$. Pas de dénormalisés.
- $E_{\min}$ et $E_{\max}$ sont choisis par l'utilisateur ($1 - 2^{30}$ et $2^{30} - 1$ par défaut).
- Valeurs spéciales : ± 0, $\pm\infty$, NaN (un seul type de NaN).
- **Arrondi correct** dans 5 modes d'arrondi : au plus près (roundTiesToEven), vers $-\infty$, vers $+\infty$, vers 0, vers l'infini de même signe (away from zero).

  Pour toutes les fonctions, dans n'importe quelle précision.

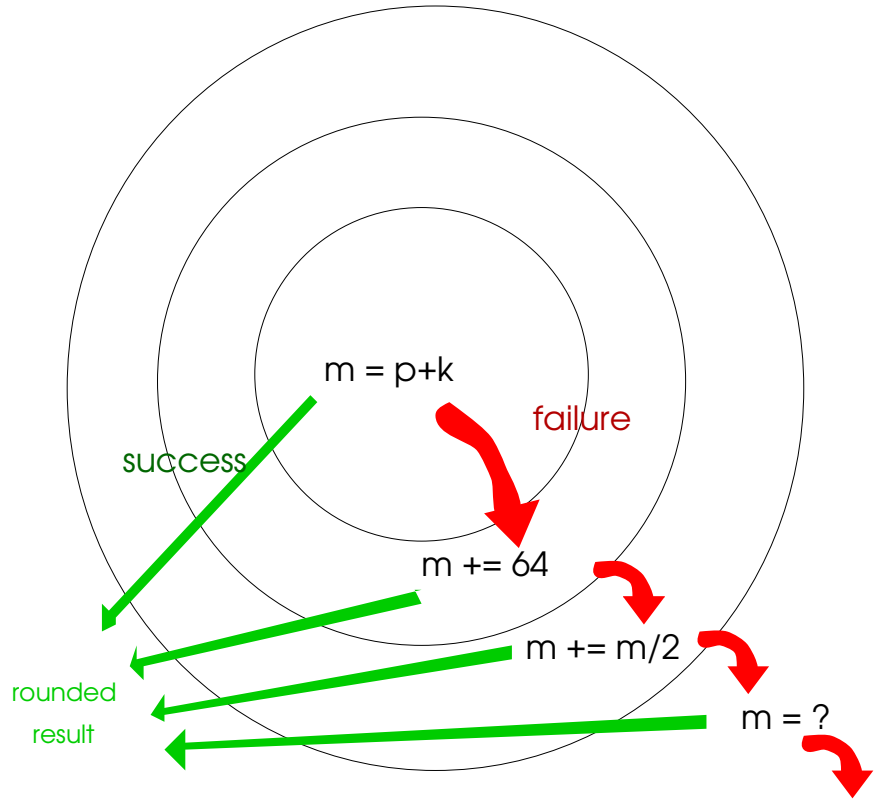
  Note : les opérandes sont toujours considérés comme exacts.

  Utilité en précision arbitraire : reproductibilité, test d'arithmétiques.

Inconvénients et difficultés

- Arrondi correct, précision non fixe, valeurs spéciales
  $\rightarrow$ coût supplémentaire visible dans de très petites précisions.

- Arrondi correct $\rightarrow$ nettement plus lent dans certains cas difficiles, normalement rares (*dilemme du fabricant de tables*).
  Stratégie de Ziv dans MPFR :
  - précision d'évaluation m un peu supérieure à la précision cible p ;
  - si arrondi correct non garanti, alors $m \leftarrow m + (32 \text{ ou } 64)$, et nouveau calcul ;
  - pour les échecs suivants : $m \leftarrow m + \lfloor m/2 \rfloor$.



- Les fonctions sin, cos et tan sur des arguments avec un très grand exposant prennent beaucoup de mémoire et sont très lentes.

Exceptions (drapeaux persistants, globaux par thread)

Invalid/NaN Le résultat MPFR n'est pas défini : NaN renvoyé.
Ex.: $0/0$; $\log(-17)$; mais aussi `mpfr_set` sur NaN.

DivideByZero Le résultat est un infini exact alors que les opérandes sont finis.
Ex.: $1/\pm 0$, qui donne $\pm\infty$; $\log(\pm 0)$, qui donne $-\infty$.

Overflow L'exposant du résultat arrondi avec exposants non bornés serait plus grand que $E_{\max}$.
Ex.: $2^{E_{\max}}$; et même `mpfr_set(y,x,MPFR_RNDU)` avec $x = \text{nextbelow}(+\infty)$ et $\text{prec}(y) < \text{prec}(x)$.

Underflow L'exposant du résultat arrondi avec exposants non bornés serait plus petit que $E_{\min}$.
Ex.: Si $E_{\min} = -17$, un underflow se produit avec `0.1e-17 / 2` et avec `0.11e-17 - 0.1e-17` (rappel : pas de dénormalisés).

Inexact Le résultat renvoyé est différent du résultat exact (car arrondi).

Erange Un résultat qui n'est pas un objet MPFR n'est pas représentable.
Ex.: `mpfr_get_ui` sur la valeur -1 ; comparaison avec NaN.

La valeur ternaire

La plupart des fonctions qui ont un nombre MPFR comme résultat (pointeur passé en premier argument) renvoient aussi un entier, appelé *valeur ternaire*.

- $= 0$ Le résultat renvoyé est exact (NaN inclus), i.e. pas d'arrondi.
- > 0 Le résultat renvoyé est plus grand que le résultat exact.
- < 0 Le résultat renvoyé est plus petit que le résultat exact.

Utilité :

- Information sur l'exactitude du résultat (mais un booléen suffirait).
- Permet de déduire le résultat dans une précision ou plage d'exposants plus petite. → Même genre d'utilisation que l'« arrondi impair » (roundToOdd), qui a été introduit dans le draft de IEEE 754-2029.

MPFR lève le drapeau *inexact* ssi la valeur ternaire est non nulle (et il reste levé jusqu'à ce que l'utilisateur le baisse).

Quelques différences entre MPFR et IEEE 754

- Pas de dénormalisés dans MPFR, mais ils peuvent être émulés avec la fonction `mpfr_subnormalize` (qui nécessite la valeur ternaire).
- La plage d'exposants (mais elle peut être fixée à celle d'un format IEEE 754).
- MPFR n'a qu'un seul NaN (similaire aux signaling NaNs pour les exceptions).
- L'exception *invalid* est un peu différente : MPFR a une exception liée à un résultat NaN et l'exception *erange*.
- La représentation mémoire est forcément différente.

Gestion de la mémoire

- Type `mpfr_t` : `typedef __mpfr_struct mpfr_t[1];`
  - quand une variable de type `mpfr_t` est déclarée, cette structure est allouée automatiquement ;
  - de la mémoire supplémentaire est nécessaire pour stocker le significande (la suite des chiffres en base 2) : `mpfr_init2` / `mpfr_clear`.
- Des nombres MPFR avec plus de précision que ce que déclare l'utilisateur peuvent être créés en interne, e.g. pour $\sin(2^{1000000000})$.
- Certaines fonctions de MPFR peuvent créer des caches, notamment pour certaines constantes comme π.
- Les données internes à MPFR (les drapeaux des exceptions, $E_{\min}$ et $E_{\max}$, les caches...) sont attachées au thread (globales si pas de multithreading). Possibilité de caches partagés par des threads (choix à la compilation de MPFR).

Test de MPFR (`make check`)

Ce qui est testé, en particulier :

- Valeurs spéciales en entrée et en sortie.
- Cas exacts ou milieux particuliers.
- Valeurs avec longues séquences de chiffres 0 ou 1.
- Cas difficiles à arrondir aléatoires lorsque la réciproque est disponible.
- Cas correspondants à des seuils d'overflow pour certaines fonctions.
- Réutilisation de variables (`reuse.c`), e.g. `mpfr_exp(x,x,rnd)`.
- Influence de données existantes avant l'appel (drapeaux, signe du résultat).
- Tests génériques de cohérence (calcul dans 2 précisions cibles).
  - Entrées aléatoires ou particulières (e.g. la plus grande valeur finie).
  - Tests avec plages d'exposant restreintes, e.g. $[17, 59]$.
  - Cohérence entre le résultat, les drapeaux et la valeur ternaire.

Compilation et exécution de `make check` dans différents contextes, e.g.
`-Wall -Werror -fsanitize=undefined -fno-sanitize-recover`

Utilisation de MPFR dans le monde

De nombreux logiciels utilisent MPFR ou en dépendent. Par exemple :

- langages/compilateurs : GCC, Julia, interfaces à divers langages ;
- arithmétique d'intervalles : MPFI, paquet *interval* de GNU Octave, Arb ;
- nombres complexes : MPC ;
- calcul formel : Mathemagix, SageMath ;
- calculateurs : GNOME Calculator, KCalc (pour KDE), Qalculate.

MPFR est utilisée pour vérifier CORE-MATH (fonctions avec arrondi correct pour les formats IEEE 754) et LLVM libc.

Publications citant MPFR :

- au moins 33 thèses ;
- Google Scholar indique 1441 publications citant l'article de référence.
  (Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, Paul Zimmermann. *MPFR : A Multiple-Precision Binary Floating-Point Library With Correct Rounding*. ACM Transactions on Mathematical Software, 2007)

Support

- Manuel aux formats info, HTML et PDF.
- Site web : `https://www.mpfr.org/` (avec manuel, FAQ, patches...).
- Dépôt Git : `https://gitlab.inria.fr/mpfr/mpfr`
- Listes de diffusion :
  - `https://sympa.inria.fr/sympa/arc/mpfr` (utilisateurs et développeurs) ;
  - `https://sympa.inria.fr/sympa/arc/mpfr-announce` (annonces).