A tutorial on reliable numerical computation

Paul Zimmermann

Dagstuhl seminar 17481, Schloß Dagstuhl, November 2017

The IEEE 754 standard

First version in 1985, first revision in 2008, another revision for 2018.

Standard *formats*: `binary32`, `binary64`, `binary128`, `decimal64`, `decimal128`

Standard *rounding modes* (attributes): `roundTowardPositive`, `roundTowardNegative`, `roundTowardZero`, `roundTiesToEven`

Correct rounding: required for $+, -, \times, \div, \sqrt{\cdot}$, recommended for other mathematical functions ($\sin, \exp, \log, \ldots$)

Different kinds of arithmetic in various languages

	fixed precision	arbitrary precision
real numbers	double (C) RDF (Sage)	GNU MPFR (C) RealField(p) (Sage)
real intervals	Boost Interval (C++) INTLAB (Matlab, Octave) RIF (Sage) RBF (Sage)	MPFI (C) RealIntervalField(p) (Sage) RealBallField(p) (Sage) libieeep1788 (C++) Octave Interval (Octave)

libieeep1788 is developed by Marco Nehmeier

GNU Octave Interval is developed by Oliver Heimlich

An example from Siegfried Rump

Reference: S.M. Rump. Gleitkommaarithmetik auf dem Prüfstand [Wie werden verifiziert(e) numerische Lösungen berechnet?]. Jahresbericht der Deutschen Mathematiker-Vereinigung, 118(3):179–226, 2016.

$$p(a, b) = 21b^2 - 2a^2 + 55b^4 - 10a^2b^2 + \frac{a}{2b}$$

Evaluate p at $a = 77617$, $b = 33096$.

Rump's polynomial in the C language

```
#include <stdio.h>

TYPE p (TYPE a, TYPE b)
{
  TYPE a2 = a * a;
  TYPE b2 = b * b;
  return 21.0*b2 - 2.0*a2 + 55*b2*b2 - 10*a2*b2 + a/(2.0*b);
}

int main()
{
  printf ("%.16Le\n", (long double) p (77617.0, 33096.0));
}
```

Rump's polynomial in the C language: results

```
$ gcc -DTYPE=float rump.c && ./a.out
-4.3870930862080000e+12

$ gcc -DTYPE=double rump.c && ./a.out
1.1726039400531787e+00

$ gcc -DTYPE="long double" rump.c && ./a.out
1.1726039400531786e+00

$ gcc -DTYPE=__float128 rump.c && ./a.out
-8.2739605994682137e-01
```

The MPFR library

A reference implementation of IEEE 754 in (binary) arbitrary precision in the C language.

Developed since 2000 by many people: Guillaume Hanrot, Fabrice Rouillier, Paul Zimmermann, Sylvie Boldo, Jean-Luc Rémy, Emmanuel Jeandel, Mathieu Dutour, Vincent Lefèvre, David Daney, Alain Delplanque, Ludovic Meunier, Patrick Pélissier, Laurent Fousse, Damien Stehlé, Philippe Théveny, Sylvain Chevillard, Charles Karney, Fredrik Johannsson, Mickaël Gastineau.

Used by GCC since 2008 for *constant folding*, to replace at compile-time expressions like `sin(1e22)` by their correctly rounded value.

Available from `mpfr.org`.

Rump's polynomial with MPFR

```
double p (double a, double b, mpfr_prec_t prec) {
  mpfr_t a2, b2, s, t; double res;
  mpfr_inits2 (prec, a2, b2, s, t, (mpfr_ptr) NULL);
  mpfr_set_d (t, a, MPFR_RNDN); mpfr_mul (a2, t, t, MPFR_RNDN);
  mpfr_set_d (t, b, MPFR_RNDN); mpfr_mul (b2, t, t, MPFR_RNDN);
  mpfr_mul_ui (s, b2, 21, MPFR_RNDN); /* 21*b^2 */
  mpfr_mul_ui (t, a2, 2, MPFR_RNDN); /* 2*a^2 */
  mpfr_sub (s, s, t, MPFR_RNDN); /* 21*b^2-2*a^2 */
  mpfr_mul (t, b2, b2, MPFR_RNDN); /* b^4 */
  mpfr_mul_ui (t, t, 55, MPFR_RNDN); /* 55*b^4 */
  mpfr_add (s, s, t, MPFR_RNDN); /* 21*b^2-2*a^2+55*b^4 */
  mpfr_mul (t, a2, b2, MPFR_RNDN); /* a2*b2 */
  mpfr_mul_ui (t, t, 10, MPFR_RNDN); /* 10*a2*b2 */
  mpfr_sub (s, s, t, MPFR_RNDN);
  mpfr_set_d (t, a, MPFR_RNDN); mpfr_div_d (t, t, b, MPFR_RNDN);
  mpfr_div_ui (t, t, 2, MPFR_RNDN); /* a/(2b) */
  mpfr_add (s, s, t, MPFR_RNDN);
  res = mpfr_get_d (s, MPFR_RNDN);
  mpfr_clears (a2, b2, s, t, (mpfr_ptr) NULL);
  return res; }
```

Rump's polynomial with MPFR

```
$ gcc rump_mpfr.c -lmpfr
$ ./a.out 24
-4.3980465111040000e+12

$ ./a.out 53
1.1726039400531787e+00

$ ./a.out 113
-8.2739605994682142e-01
```

Note: only 17 significant digits are printed since the function `p` returns a double. If we print the MPFR variable `s` before converting it to double, with

```
  mpfr_printf ("s=%Re\n", s);
```

then we get:

```
$ ./a.out 24
s=-4.39804651e+12

$ ./a.out 53
s=1.172603940053178 7e+00

$ ./a.out 113
s=-8.27396059946821368141165095479816202e-01
```

Using the SageMath system

SageMath (Sage for short) is a computer algebra system written in the Python language.

It calls several dozens of specialized libraries (Pari/GP, Singular, NTL, Maxima, Sympy, ...)

It is an open-source software, available from `sagemath.org`, with more than 600 developers.

Now in english!

`http://www.loria.fr/~zimmerma/sagebook/english.html`

Floating-Point Numbers in Sage

- `RealDoubleField` or `RDF`: machine double precision
- `RealField(p)` for $p \geq 2$: p-bit precision (using the MPFR library)
- `RealIntervalField(p)`: p-bit inf-sup interval arithmetic (using the MPFI library)
- `RealBallField(p)`: p-bit mid-rad interval arithmetic (using the Arb library developed by Fredrik Johansson)

Rump's polynomial with machine double precision in Sage

```
SageMath version 8.0, Release Date: 2017-07-21
Type "notebook()" for the browser-based notebook interface.
Type "help()" for help.

sage: def p(x,y):
....:     return 21*y^2-2*x^2+55*y^4-10*x^2*y^2+x/(2*y)

sage: a = RDF(77617); b = RDF(33096)
sage: p(a,b)
1.172603940053178?
```

The RealField class

`R = RealField(17)` creates the set of 17-bit (binary) floating-point numbers. It is a wrapper above the MPFR library.

By default rounding is to nearest. `RealField(17,rnd='RNDZ')` will round toward zero instead.

Once an object of this domain has been created, for example with `a = R(pi)`, several methods are available for the object `a`:

```
sage: R = RealField(17)
sage: a = R(pi)
sage: a.exact_rational()
3217/1024
sage: a.sign_mantissa_exponent()
(1, 102944, -15)
sage: s,m,e = a.sign_mantissa_exponent()
sage: a.exact_rational() == s*m*2^e
True
```

The toy field RealField(2)

```
sage: R2 = RealField(2)
sage: x = R2(1/pi)
sage: x
0.38
sage: x.ulp()
0.12
sage: x.ulp().exact_rational()
1/8
sage: x.exact_rational()
3/8
sage: x.nextabove()
0.50
sage: x.nextbelow()
0.25
```

Rump's polynomial with RealField in Sage

```
sage: def p(x,y):
....:     return 21*y^2-2*x^2+55*y^4-10*x^2*y^2+x/(2*y)

sage: R = RealField(53)
sage: a, b = R(77617), R(33096)
sage: p(a,b)
1.17260394005318

sage: R = RealField(113)
sage: a, b = R(77617), R(33096)
sage: p(a,b)
-0.82739605994682136814116509547981
```

The IEEE 1788 standard

A standard for interval arithmetic, first version in 2015.

Two main representations: inf-sup and mid-rad.

With the inf-sup representation, $[\ell, h]$ represents $\ell \leq x \leq h$

With the mid-rad representation, (m, r) represents $m - r \leq x \leq m + r$

Hopefully soon we have corresponding native types in the C language.

The MPFI library

An arbitrary-precision interval arithmetic library.

inf-sup encoding: $[\ell, h]$ represents all numbers $\ell \leq x \leq h$.

ℓ and h are floating-point numbers, represented using MPFR.

Developed by Nathalie Revol, Fabrice Rouillier, Sylvain Chevillard, Christoph Lauter, Hong Diep Nguyen, Philippe Théveny.

Available from `http://mpfi.gforge.inria.fr/`

Rump's polynomial with RealIntervalField in Sage

```
sage: def p(x,y):
....:     return 21*y^2-2*x^2+55*y^4-10*x^2*y^2+x/(2*y)

sage: R = RealIntervalField(53)
sage: a, b = R(77617), R(33096)
sage: p(a,b)
0.?e5
sage: sage.rings.real_mpfi.printing_style = 'brackets'
sage: p(a,b)
[-8190.8273960599473 .. 16385.172603940057]

sage: R = RealIntervalField(64)
sage: a, b = R(77617), R(33096)
sage: p(a,b)
[-2.8273960599468213681 .. 1.7260394005317863194]

sage: R = RealIntervalField(65)
sage: a, b = R(77617), R(33096)
sage: p(a,b)
[-0.82739605994682136814 .. -0.82739605994682136809]
```

The Arb library

An interval arithmetic library using the mid-rad encoding.

(m, r) represents the interval $m - r \leq x \leq m + r$

m and r are floating-point numbers, but usually a small precision is enough for r

Developed by Fredrik Johansson.

Available from `arblib.org`

Rump's polynomial with RealBallField in Sage

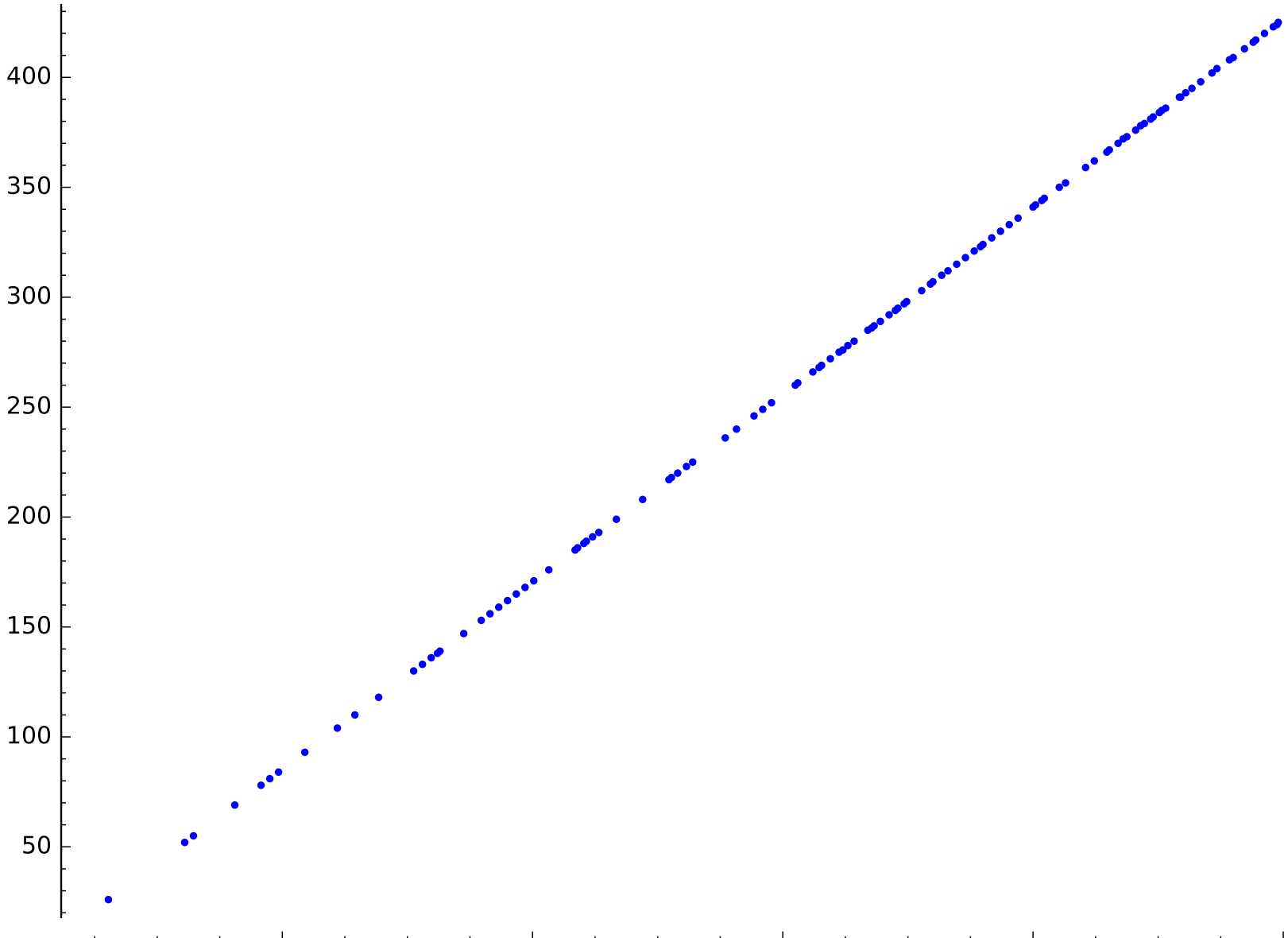
```
sage: def p(x,y):
....:     return 21*y^2-2*x^2+55*y^4-10*x^2*y^2+x/(2*y)

sage: R = RealBallField(53)
sage: a, b = R(77617), R(33096)
sage: p(a,b)
[+/- 2.46e+4]

sage: R = RealBallField(64)
sage: a, b = R(77617), R(33096)
sage: p(a,b)
[+/- 6.83]

sage: R = RealBallField(65)
sage: a, b = R(77617), R(33096)
sage: p(a,b)
[-0.827396059946821368 +/- 7.81e-20]
```

We can look at which integers $1 \leq a, b \leq 1000$ the sign of $p(a, b)$ computed with `RealField(10)` is incorrect (a, b can be exactly represented with 10 bits):



Conclusion

Different kinds of arbitrary-precision arithmetic in various languages

For ease of use and rigorous bounds, use `RealIntervalField` or `RealBallField` within Sage

For efficiency, use MPFI or Arb (or MPFR if you can work out the error bounds with paper and pencil)