Exact real arithmetic

Keith Briggs

Keith.Briggs@bt.com

more.btexact.com/people/briggsk2/XR.html

BT exact TECHNOLOGIES

2002 Nov 20 15:00

Typeset in LaTeX2e on a linux system

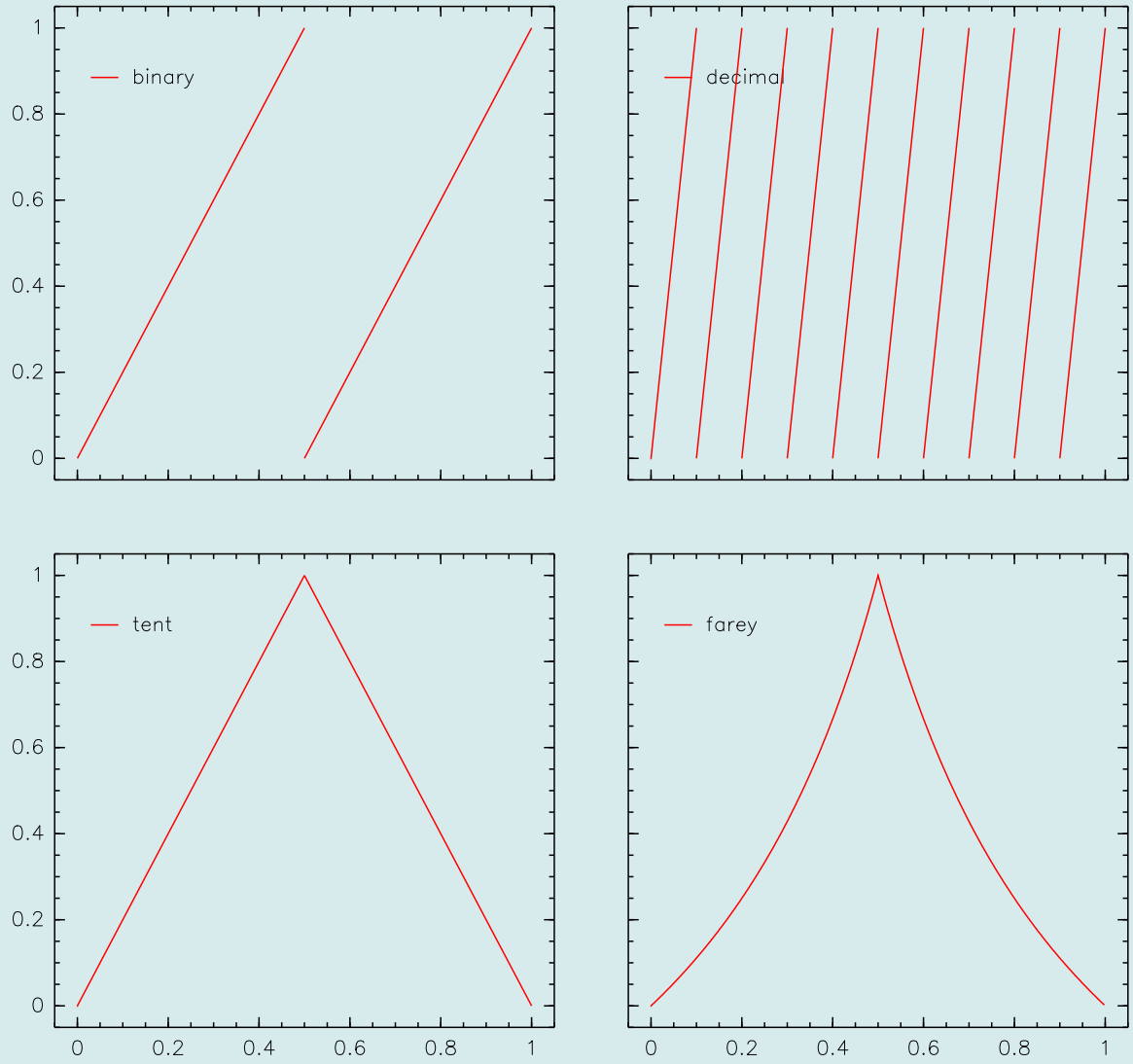
Outline

- What's the problem?
- Representation of reals
- Processing of reals
- Applications

Real numbers

- Integers: $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
- Rationals: $\mathbb{Q} = \{p/q;\ p, q \in \mathbb{Z}\}$
- Irrationals: all the rest: $\sqrt{2}, e, \pi, \ldots$
- $\mathbb{R} = \mathbb{Q} \cup$ irrationals
- $\mathbb{R} \backslash \mathbb{Q} = \{\text{algebraics} \cup \text{transcendentals}\}$

Digit expansions



$$\mathbb{F} = \{ m\,2^e : |m| < 2^{53}, |e| < 1024, \text{ NaN, Inf} \}$$
$$\mathbb{O} = \{ +, -, *, / \}$$

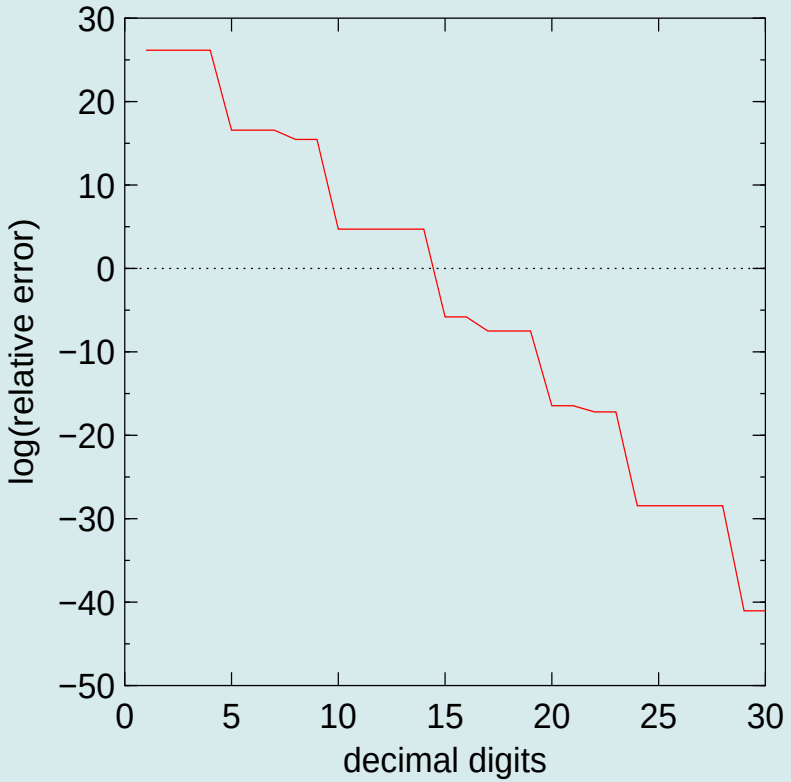
Problems!

$$x_0 = 0.9$$
$$x_{k+1} = 3.999\, x_k\, (1 - x_k) \qquad k = 0, 1, 2, \ldots$$

In $\mathbb{F}$, $x_{53} > 0.5$, but the exact result is
$x_{53} = 0.13023587481177373303964373008057 0 \ldots$

Another floating point disaster

$$u_0 = e - 1$$
$$u_k = k\,u_{k-1} - 1 \qquad k = 1, \ldots, 25$$



$$x = [x_0; x_1, x_2, x_3, \ldots] = x_0 + \cfrac{1}{x_1 + \cfrac{1}{x_2 + \cdots}}$$

Studied by Gosper, Vuillemin, Liardet, Stambul, Ménissier-Morain, Potts.

Only easy operation: $\frac{ax+b}{cx+d}$.

For $+, *$ etc., inefficient.

For exp, sin etc., very inefficient!

How do we convert inputs to required cf form?

necessarily involves a *limit* of a sequence of rationals:

- digit expansions (possibly with negative digits)
- symbolic dynamics of an expanding map
- continued fractions
- nested intervals with rational endpoints
- Dedekind cuts
- Conway's construction
- . . .

Traditional digit expansions converge strictly from below, so . . .

- Digit expansions with negative digits
- Continued fractions
- Möbius maps (LFTs)
- Non-integer base - e.g. golden ratio
- Does something even better exist? . . .

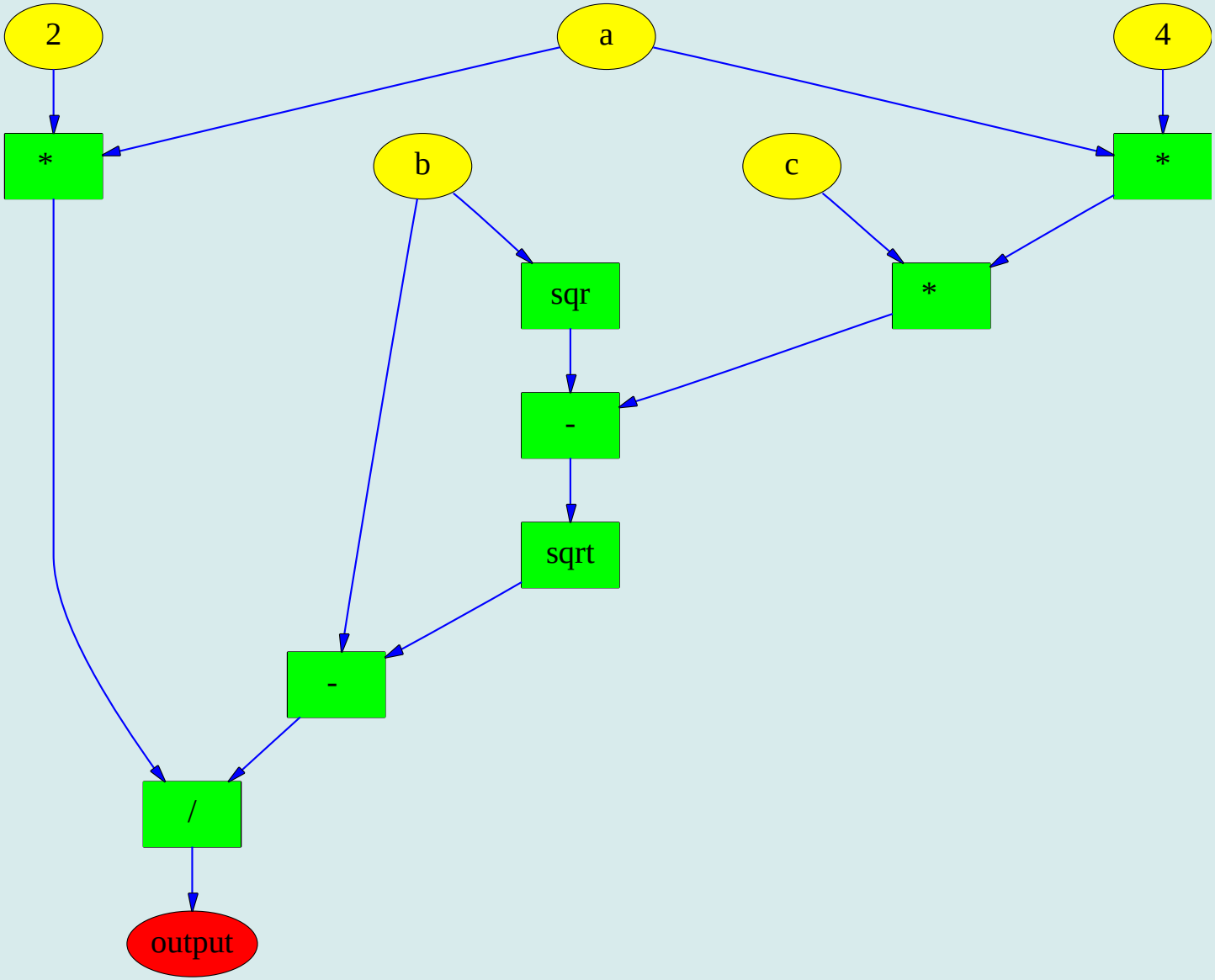
We want a representation that allows the computation of more significant digits first

Can we compute with *limits*?

Conventionally, no; but what if all our limits converge at the same rate?

- Example: solve a quadratic
- $ax^2 + bx + c = 0$
- $x = \frac{-b+\sqrt{b^2-4ac}}{2a}$

Data flow 2



What's wrong with that?

- Input nodes don't know how much precision to send
- All input nodes send data, even if it eventually may not be needed
- To recalculate requires the whole tree to be re-evaluated

For $\hat{x} \in \mathbb{R}$, consider a function χ such that

$$\boxed{| B^n \hat{x} - \chi(n) | < 1}$$

for fixed $B = 2, 3, 4, \ldots$

$n \in \mathbb{Z}^+$

$\chi : \mathbb{Z}^+ \to \mathbb{Z}$ $(\chi \in \mathbb{XR})$

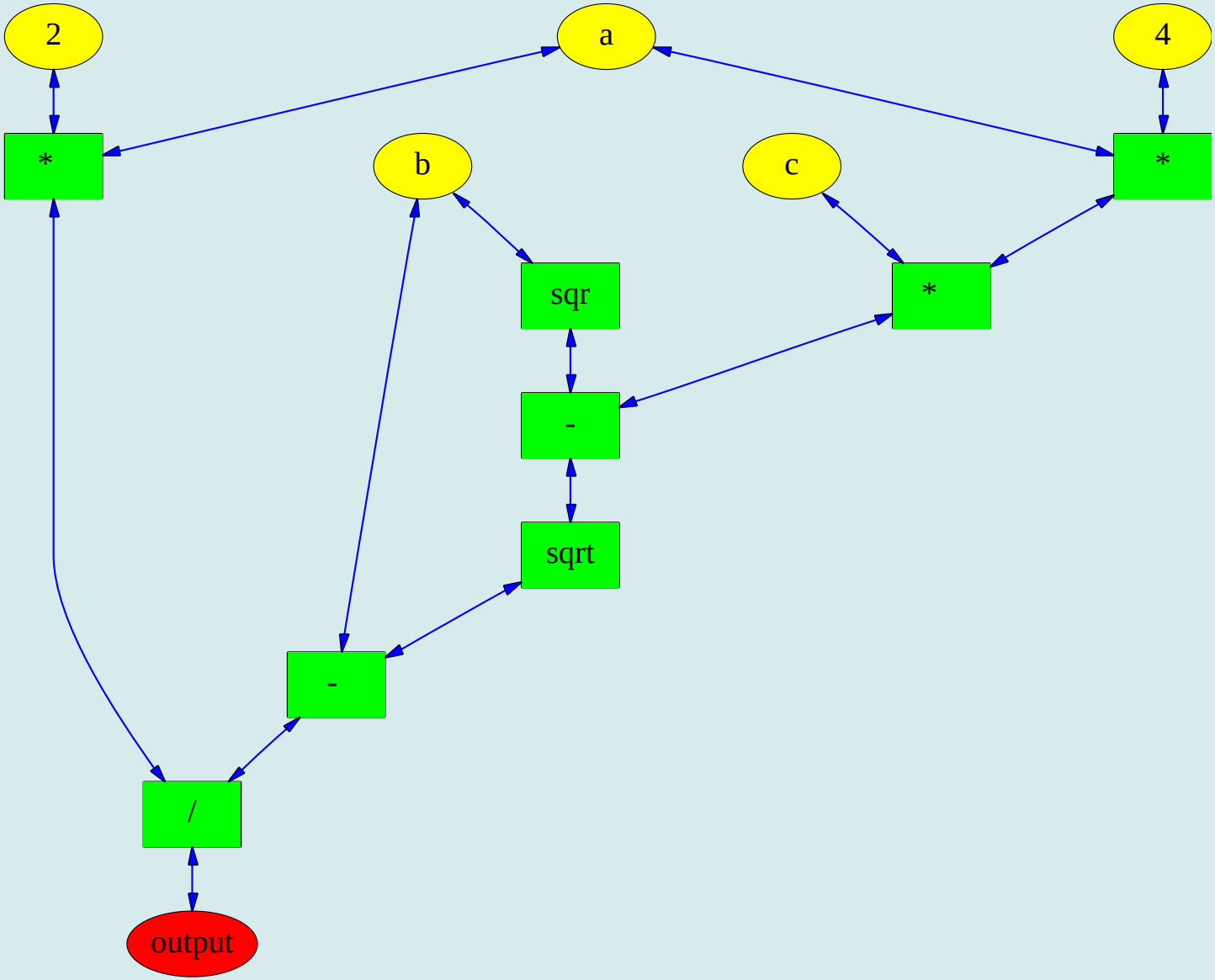
In other words,

$$\frac{x(n)-1}{B^n} < \widehat{x} < \frac{x(n)+1}{B^n}$$

Example: $\widehat{x} \in \mathbb{Q}$, $x(n) \equiv \lfloor B^n \widehat{x} \rfloor$

NB: we reduce all computation with $\mathbb{XR}$s to integer calculations.

Data flow 3



$$[x+y](n) \equiv \lfloor\, (x(n+2)+y(n+2)+2)/4 \,\rfloor$$

$$\begin{aligned}
& |\,[x+y](n) - 2^n(\widehat{x}+\widehat{y})\,| \\
= \quad & |\,\lfloor\, (x(n+2)+y(n+2)+2\,)/4\rfloor - 2^n(\widehat{x}+\widehat{y})\,| \\
\leqslant \quad & 1/2 + |\,(x(n+2)+y(n+2))/4 - 2^n(\widehat{x}+\widehat{y})\,| \\
= \quad & 1/2 + |\,x(n+2)+y(n+2) - 2^{n+2}(\widehat{x}+\widehat{y})\,|\,/\,4 \\
\leqslant \quad & 1/2 + |\,x(n+2) - 2^{n+2}\,\widehat{x}\,|\,/\,4 \\
& \qquad + |\,y(n+2) - 2^{n+2}\,\widehat{y}\,|\,/\,4 \\
= \quad & 1
\end{aligned}$$

Produce efficient software which hides the bottom-up data flow and can be used by a programmer as if it were conventional top-down code

Not possible in Fortran, C, Java, . . .
Possible in python, Haskell, clean, ML, ocaml, (and C++)

The answer: lazy evaluation

Given a polynomial p with integer coefficients, and integers $a, k > 0$, we can compute the sign of p at a/B^k with only integer operations.

```
def sign_at(p,a,k):
  n,w=len(p),p[0]
  for j in 1..n:
    w*=a
    w+=p[j]*B^(k*j)
  return w>0
```

Thus, given a bracketed root:

$$\operatorname{sign} p(a/B^k) < 0, \quad \operatorname{sign} p(b/B^k) > 0$$

we may refine it by bisection to any desired accuracy.

$f = \exp, \arctan, \sin$ etc. can be computed if we can implement an approximating function $\tilde{f} : \mathbb{Q} \times \mathbb{Z}^+ \to \mathbb{Z}$ such that

$$\left| B^n f(q) - \tilde{f}(q, n) \right| < 1$$

This can be achieved using Taylor polynomials with bounds on the error term.
For π, use

$$\pi = \sum_{n=0}^{\infty} \left[\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right] \frac{1}{16^n}$$

`www.python.org`

- ‘improved perl’
- much cleaner syntax than java . . .
- . . . but just as powerful
- portable
- functional features - lambda
- operator overloading
- free!

Lambdas

```
def +(x,y):
  return lambda n: (x(n+2)+y(n+2)+2)/4

def sqrt(x):
  return lambda n: sqrt(x(2*n))
```

Principle:

Do not evaluate arguments until necessary

```
def if_nonzero(f,x):
    " return a function which computes
   f if x is nonzero, else None "
    if x: return lambda x: f(x)
    return None
```

Equality is undecidable, but . . .

This is a system for proving inequalities

```
def <(x,y):
  n=1
  while 1:
    if x(n)<y(n)-1: return true
    if x(n)>y(n)+1: return false
    n++
```

Python implementation in action

```
kbriggs@tontolum:~/XR-2.0> python
Python 2.2 (#1, Dec 31 2001, 13:51:20)
[GCC 2.95.2 19991024 (release)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from XR import *
>>> x=exp(pi()*sqrt(XR(163)))
>>> print floor(x)<x
1
>>> e=exp(XR(1))
>>> print contfrac(e)
[2L, 1L, 2L, 1L, 1L, 4L, 1L, 1L, 6L, 1L, 1L, 8L, 1L, 1L, 10L, 1L, 1L, 12L, 1L, 1
L, 14L]
>>> print cos(2*pi()/7).dec(50)
 0.62348980185873353052500488400423981063227473089640e0
>>> print polyroot([8,4,-4,-1],0,1,0).dec(50)
 0.62348980185873353052500488400423981063227473089640e0
>>> x=XR(Q(1,3))
>>> for i in range(20):
...     print 2*x>1,
...     x=4*x*(1-x)
...
0 1 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1 1 1 0
>>>
```

Python implementation in action

```
> import XR
> x=exp(pi()*sqrt(XR(163)))
> print floor(x)<x
1
> e=exp(XR(1))
> print contfrac(e)
[2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1,
1, 12, 1, 1, 14]
> print cos(2*pi()/7).dec(50)
 0.62348980185873353052500488400423981063227473
> print polyroot([8,4,-4,-1],0,1,0).dec(50)
 0.62348980185873353052500488400423981063227473
> x=XR(Q(1,3))
> for i in range(10):
...     print 2*x>1
...     x=4*x*(1-x)
...
0 1 0 1 0 1 1 0 0 1
```

C++ implementation

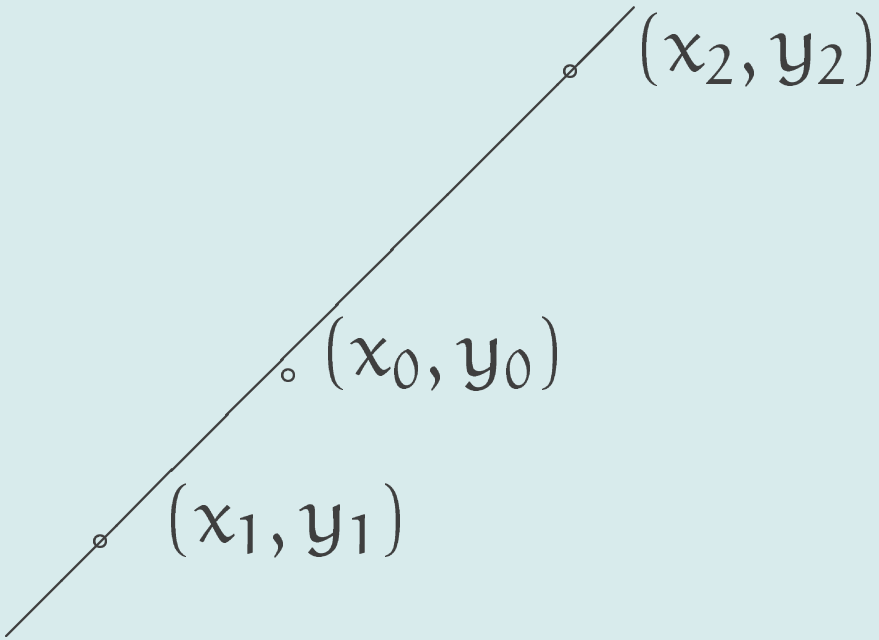
```
#include "XR.h"

int main() {
  XR x=exp(pi()*sqrt(XR(163)));
  cout<<floor(x)<x;
  XR e=exp(XR(1));
  cout<<contfrac(e);
  cout<<cos(2*pi()/7).dec(50);
  ZZ coeffs[]={8,4,-4,-1};
  cout<<polyroot(coeffs,0,1,0).dec(50);
  x=XR(Q(1,3));
  for (int i=0; i<10; i++) {
    cout<<2*x>1;
    x=4*x*(1-x);
  }
  return 0;
}
```

Features of my implementations

- python, C++
- All integer arithmetic
- Caching
- Easily integrated with existing code
- C++ version is fully compiled
- Applications in computational number theory, computer-assisted proofs, computer algebra, . . .

Simple planar example:
Line through $(x_1, y_1), (x_2, y_2)$; is the point (x_0, y_0) to the left or right of the line?



This is determined by

$$\text{sign}\left([(y_1 - y_0)(x_2 - x_1) - (x_1 - x_0)(y_2 - y_1)]\right)$$

With $\mathbb{XR}$ arithmetic, the sign is always determined correctly and with the minimal necessary computation

1. Simultaneous Diophantine approximation algorithms
   Typical subproblem: given
   $x_1, x_2 \in \mathbb{XR}$
   $p_1, p_2, q \in \mathbb{Z}$
   $e_1 = |qx_1 - p_1|$, $e_2 = |qx_2 - p_2|$
   Is $e_1 > e_2$ or not?
2. Is $\exp(\pi\sqrt{163})$ an integer?

```
x=exp(pi*sqrt(XR(163)))
f=floor(x)
print x>f, x<f+1 # => true, true
```

Problems

1. Memory demands
2. Non-incrementality
3. The floor function does not terminate on integer imputs
4. This makes implementing exp etc. hard
5. (Verified) decimal output

Mathematics $\Longrightarrow$ mathematical software

- Numerical analysis
- Statistics
- Computer algebra
- Computational number theory
- Combinatorics and graph theory
- Finite groups
- Theorem proving
- . . .

What about the other direction?

It’s worth rethinking how we represent numbers and do arithmetic, especially for distributed applications

V Ménissier-Morain: *Arithmétique exacte, conception, algorithmique et performances d'une implémentation informatique en précision arbitraire*
Paris thesis 1994
`calfor.lip6.fr/~vmm/`

J R Harrison: *Theorem proving with the real numbers*
Cambridge thesis 1996
`www.ftp.cl.cam.ac.uk/ftp/papers/reports/`

K M Briggs: *XR homepage*
`more.btexact.com/people/briggsk2/XR.html`