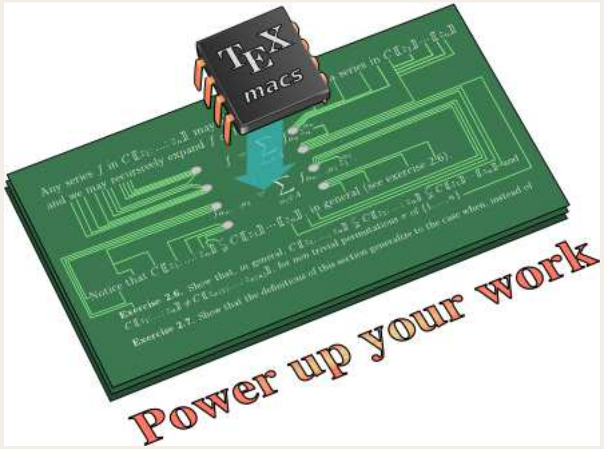
Effective real numbers

by Joris van der Hoeven



Presentation with GNU TEXmacs (`www.texmacs.org`)

Effective real numbers in MMXLIB

Mmx $\gg$ x: Real $==$ sin sin real 2;

Mmx $\gg$ x

$7.891\,10^{-1}$ Real

Mmx $\gg$ approximate(x, 1.0e−35);

$7.8907234357288836143140304248688412\,10^{-1}$ Interval

Mmx $\gg$ M: Matrix Real $== \begin{pmatrix} x & x+2 \\ 2-x^2 & \cos x \end{pmatrix}$;

Mmx $\gg$ M;

$$\begin{bmatrix} 7.891\,10^{-1} & 2.789 \\ 1.377 & 7.045\,10^{-1} \end{bmatrix}$$ Matrix(Real)

Mmx $\gg$ M^{20};

$$\begin{bmatrix} 2.284\,10^{8} & 3.181\,10^{8} \\ 1.571\,10^{8} & 2.188\,10^{8} \end{bmatrix}$$ Matrix(Real)

Mmx $\gg$ $\exp(x + \exp(-\text{real } 100)) - \exp(x)$;

$8.189\,10^{-44}$ Real

Mmx $\gg$ $\exp(x + \exp(-\text{real } 1000)) - \exp(x)$;

0 Real

Effective analytic functions in MMXLIB

Mmx $\gg$ z: Analytic == analytic $(0, 1)$;

Mmx $\gg$ $\exp(z)$;

$1.000 + 1.000\ z + 5.000\ 10^{-1}\ z^2 + 1.667\ 10^{-1}\ z^3 + 4.167\ 10^{-2}\ z^4 + 8.333\ 10^{-3}\ z^5 + 1.389\ 10^{-3}\ z^6 + 1.984\ 10^{-4}\ z^7 + 2.480\ 10^{-5}\ z^8 + 2.756\ 10^{-6}\ z^9 + O(z^{10})$ Analytic

Mmx $\gg$ $\exp(z)$[int 20];

$4.110\ 10^{-19}$ Complex

Mmx $\gg$ ℓ: Analytic == $\log(1 - z)$;

Mmx $\gg$ radius(ℓ);

$9.99937726184725761359\ 10^{-1}$ Floating

Mmx $\gg$ evaluate$(\ell$, complex$(1/2))$;

$-6.931\ 10^{-1}$ Complex

Mmx $\gg$ continuate$(\ell$, complex$(1/2))$;

$-6.931\ 10^{-1} - 2.000\ z - 2.000\ z^2 - 2.667\ z^3 - 4.000\ z^4 - 6.400\ z^5 - 1.067\ 10^1\ z^6 - 1.829\ 10^1\ z^7 - 3.200\ 10^1\ z^8 - 5.689\ 10^1\ z^9 + O(z^{10})$ Analytic

Mmx $\gg$ continuate$(\ell$, turn(complex$(1)))$;

$6.283\ \mathrm{i} - 1.000\ z - 5.000\ 10^{-1}\ z^2 - 3.333\ 10^{-1}\ z^3 - 2.500\ 10^{-1}\ z^4 - 2.000\ 10^{-1}\ z^5 - 1.667\ 10^{-1}\ z^6 - 1.429\ 10^{-1}\ z^7 - 1.250\ 10^{-1}\ z^8 - 1.111\ 10^{-1}\ z^9 + O(z^{10})$ Analytic

Mmx ≫

Solving differential equations

$$f'' = (z^2 + 1)\ f' + \mathrm{e}^z\ f; \qquad f(0) = 1,\ f'(0) = 1 + 2\,\mathrm{i}.$$

Mmx ≫ $f\colon \mathrm{Analytic} == \mathrm{solve\_lde}((z^2+1, \exp(z)), (\mathrm{complex}(1), \mathrm{complex}(1,2)));$

Mmx ≫ $f;$

$1.000 + (1.000 + 2.000\ \mathrm{i})\ z + (1.000 + 1.000\ \mathrm{i})\ z^2 + (6.667\ 10^{-1} + 1.000\ \mathrm{i})\ z^3 + (5.417\ 10^{-1} + 5.833\ 10^{-1}\ \mathrm{i})\ z^4 + (3.500\ 10^{-1} + 4.833\ 10^{-1}\ \mathrm{i})\ z^5 + (2.278\ 10^{-1} + 2.750\ 10^{-1}\ \mathrm{i})\ z^6 + (1.343\ 10^{-1} + 1.742\ 10^{-1}\ \mathrm{i})\ z^7 + (7.882\ 10^{-2} + 9.767\ 10^{-2}\ \mathrm{i})\ z^8 + (4.361\ 10^{-2} + 5.572\ 10^{-2}\ \mathrm{i})\ z^9 + O(z^{10})$ Analytic

Mmx ≫ $u\colon \mathrm{Complex} == \mathrm{evaluate}(f, \mathrm{complex}(1/10));$

Mmx ≫ $u;$

$1.111 + 2.111\ 10^{-1}\ \mathrm{i}$ Complex

Mmx ≫ $\mathrm{approximate}(u, 1.0e{-}81);$

$1.11072457537794457102292725574830566357052541308848196626687047567517902828392834 + 2.1106346012282867466007605052618438398248510727864880851400427655460836641117663\ 10^{-1}\ \mathrm{i}$
Complexify(Interval)

Mmx ≫

Definition of effective real numbers

- $\tilde{x} \in \mathbb{D} = \mathbb{Z}\, 2^{\mathbb{Z}}$ is an ε-approximation of $x \in \mathbb{R}$ if $|\tilde{x} - x| < \varepsilon$.
- *Approximation algorithm* for x: computes $\varepsilon \longmapsto \varepsilon$-approximation of x.
- *Effective real number*: $x \in \mathbb{R}$ which admits an approximation algorithm.
- *Complexity* of x: time needed to compute a 2^{-l}-approximation.
- Implementation of `real` as pointer to

```
class real_rep {
public:
  virtual dyadic approximate (const dyadic& err) = 0;
  ...
};
```

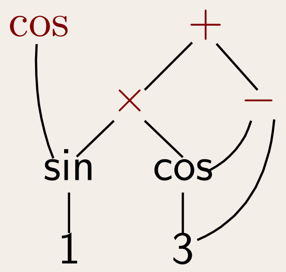
- No zero-test for effective real numbers.
- References: Bishop and Bridges, Blanck, Müller, vdH, etc.

Modeling computations with effective real numbers

- Example: addition

```
class add_real_rep: public real_rep {
  real x, y;
  add_real_rep (const real& x2, const real& y2):
    x (x2), y (y2) {}
  dyadic approximate (const dyadic& eps) {
    return x->approximate (eps/2) + y->approximate (eps/2); }
};
```

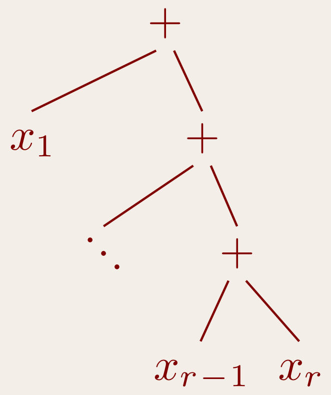
- Model sets of effective real numbers by acyclic graphs:

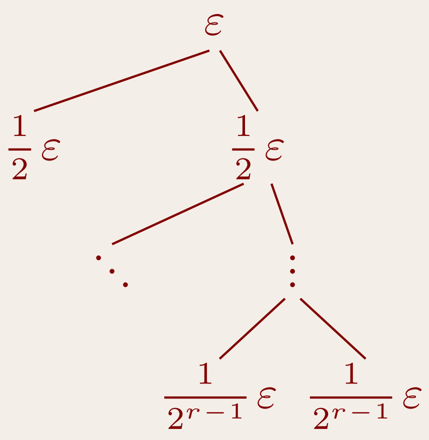


- Computations stored in memory $\rightarrow$ don't use classical numerical algorithms.

A priori error estimates

- Distribute tolerance ε *a priori* over nodes of n-ary operations ($n > 1$).
- Can be bad in case of badly nested expressions:

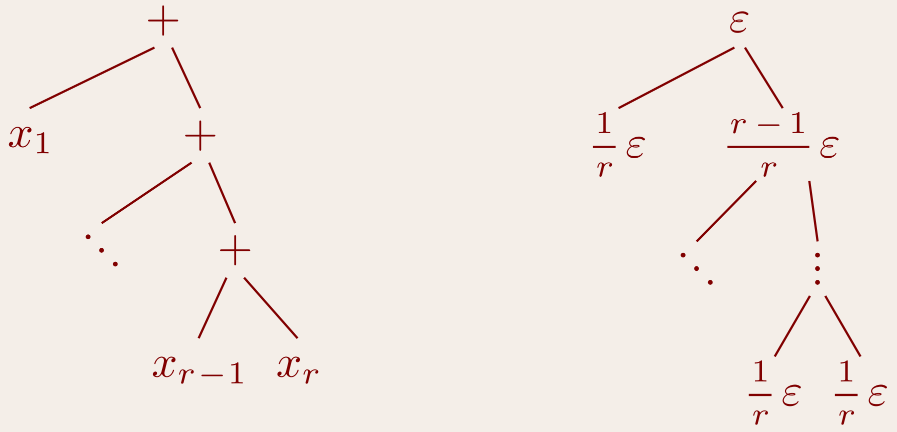




- Possible loss of $\log w$ bits of precision; still unacceptable.

A priori error estimates

- Distribute tolerance ε *a priori* over nodes of n-ary operations ($n > 1$).
- Balanced error estimates: redistribute as a function of *weight*:



- Possible loss of $\log w$ bits of precision; still unacceptable.

A posteriori error estimates

- Perform whole computation using interval arithmetic.

  While result not precise enough:

    Double precision and redo entire computation.

- First improvement:

  For each instance of `real_rep`, keep best current approximation in memory.

- Second improvement:

  Don't double precision, but estimated computation time.

Representation of intervals

- $W = 16, 32, 64, 128, \ldots$ bit precision of the processor.
- $\mathbb{D}_l$ set of generalized IEEE 754 numbers of precision l and fixed exponent range.
- Standard representation at precision l

  $\boldsymbol{x} = [\underline{\boldsymbol{x}}, \overline{\boldsymbol{x}}]$, with $\underline{\boldsymbol{x}} \leqslant \overline{\boldsymbol{x}} \in \mathbb{D}_l \backslash \{\mathrm{NaN}\}$ or $\boldsymbol{x} = \mathrm{NaIn}$.

  - Easy to implement on top of MPFR.
  - Inherits standardization from IEEE 754.
  - Requires two l-bit computations for each operation.

Representation of intervals

- $W = 16, 32, 64, 128, \ldots$ bit precision of the processor.
- $\mathbb{D}_l$ set of generalized IEEE 754 numbers of precision l and fixed exponent range.
- Ball representation at precision l

$$\boldsymbol{x} = \mathcal{B}(c_{\boldsymbol{x}}, r_{\boldsymbol{x}}) \text{ with } \begin{cases} c_x \in \mathbb{D}_l \notin \{\mathrm{NaN}, \pm 0, \pm\infty\} \\ r_x \in \mathbb{D}_l^{\geqslant}, l > W \Rightarrow r_{\boldsymbol{x}} \leqslant |c_{\boldsymbol{x}}| 2^{W-l} \end{cases} \text{ or } \boldsymbol{x} = \mathrm{NaIn}$$

  - Efficient and easy to implement for high precisions.
  - Less expressive power: cannot represent $[0, \infty]$.
  - Difficult to preserve positivity: $\mathcal{B}(a, a) + B(b, b)$.

Representation of intervals

- $W = 16, 32, 64, 128, \ldots$ bit precision of the processor.
- $\mathbb{D}_l$ set of generalized IEEE 754 numbers of precision l and fixed exponent range.
- Hybrid representation at precision l

  Use standard representation for $l = W$.

  Use ball representation for $l > W$.

Semantics and standardization

- Precision loss and normalization

  Example: $2.0060000000123 - 2.0060000000000 \longrightarrow 1.23 \times 10^{-10}$.

  $\longrightarrow$ Set of intervals at precision l not stable under $+$, $-$, $\times$, etc.

  $\longrightarrow$ Set $\mathbb{I}_l$ of intervals at precision $\leqslant l$ is stable under $+$, $-$, $\times$, etc.

- Precision gains

  Example 1: $\log_{10} 1.0 \times 10^{1000} \longrightarrow 1.0000 \times 10^3$.

  Example 2: $\arctan 1.0 \times 10^{10} \longrightarrow 1.5707963267$.

  $\longrightarrow$ We compute with precision $\min\,(l, \mathrm{precision\,arguments})$.

- Semantics

  $\longrightarrow$ Perform operation as if we use l bit precision in standard representation.

  $\longrightarrow$ Normalize result to hybrid representation.

  $\longrightarrow$ Return NaIn as soon as error occurs for one possible value.

  $\longrightarrow$ Loosen: allow for 2^{-W} relative errors.

Template types over `interval`

- Complex numbers

Mmx $\gg$ $\mathrm{forall}(T)\ \mathrm{power}(x\colon T, n\colon \mathrm{Integer})\colon T ==$
$(\mathrm{if}\ n = 1\ \mathrm{then}\ x\ \mathrm{else}\ x\ \mathrm{power}(x, n-1));$

Mmx $\gg$ $\alpha\colon \mathrm{Interval} == \mathrm{approximate}(\mathrm{real}\ 2, 1.0e{-}24);\ \alpha;$

$2.00000000000000000000000$ Interval

Mmx $\gg$ $\beta\colon \mathrm{Complexify\ Interval} == \mathrm{approximate}(\mathrm{complex}(1,1), 1.0e{-}24);\ \beta;$

$1.00000000000000000000000 + 1.00000000000000000000000\mathrm{i}$ Complexify(Interval)

Mmx $\gg$ $\mathrm{for}\ i\colon \mathrm{Integer\ in}\ 1\ldots10\ \mathrm{loop\ mmout} \ll \mathrm{power}(\beta, 8\,i) \ll \text{"}\backslash n\text{"};$

$1.6000000000000000000000\ 10^{1} + 0.0\ 10^{-23}\ \mathrm{i}$
$2.560000000000000000000\ 10^{2} + 0.0\ 10^{-21}\ \mathrm{i}$
$4.0960000000000000000\ 10^{3} + 0.0\ 10^{-19}\ \mathrm{i}$
$6.5536000000000000000\ 10^{4} + 0.0\ 10^{-16}\ \mathrm{i}$
$1.04857600000000000\ 10^{6} + 0.0\ 10^{-14}\ \mathrm{i}$
$1.6777216000000000\ 10^{7} + 0.0\ 10^{-11}\ \mathrm{i}$
$2.68435456000000\ 10^{8} + 0.0\ 10^{-9}\ \mathrm{i}$
$4.2949672960000\ 10^{9} + 0.0\ 10^{-7}\ \mathrm{i}$
$6.871947673600\ 10^{10} + 0.0\ 10^{-4}\ \mathrm{i}$
$1.09951162778\ 10^{12} + 0.0\ 10^{-2}\ \mathrm{i}$

Mmx $\gg$ $\mathrm{for}\ i\colon \mathrm{Integer\ in}\ 1\ldots10\ \mathrm{loop\ mmout} \ll \beta^{8i} \ll \text{"}\backslash n\text{"};$

$1.600000000000000000000\,10^{1} + 0.0\,10^{-24}\,\mathrm{i}$
$2.560000000000000000000\,10^{2} + 0.0\,10^{-23}\,\mathrm{i}$
$4.096000000000000000000\,10^{3} + 0.0\,10^{-21}\,\mathrm{i}$
$6.553600000000000000000\,10^{4} + 0.0\,10^{-20}\,\mathrm{i}$
$1.048576000000000000000\,10^{6} + 0.0\,10^{-18}\,\mathrm{i}$
$1.677721600000000000000\,10^{7} + 0.0\,10^{-17}\,\mathrm{i}$
$2.684354560000000000000\,10^{8} + 0.0\,10^{-16}\,\mathrm{i}$
$4.294967296000000000000\,10^{9} + 0.0\,10^{-15}\,\mathrm{i}$
$6.871947673600000000000\,10^{10} + 0.0\,10^{-13}\,\mathrm{i}$
$1.099511627776000000000\,10^{12} + 0.0\,10^{-12}\,\mathrm{i}$

```
Mmx ≫
```

Systematically use ball representation.

- Matrices

  Compute matrix products $M_1 \cdots M_n$ by dichotomy.

  For instance: $M_1 \cdots M_8 = (((M_1\, M_2)\,(M_3\, M_4))\,((M_5\, M_6)\,(M_7\, M_8)))$

- Truncated power series

  First renormalize $f(z) \mapsto f(\rho\, z)$.

Implementing effective real numbers I

Representation and main methods

```
class real_rep {
protected:
  double   cost;
  interval best;

public:
  virtual interval compute () = 0;
  virtual int precision_for (double cost) = 0;

  real_rep (): cost (1.0), best (compute ()) {}
  interval improve (double new_cost);
  interval approximate (const dyadic& err);
};
```

Implementing effective real numbers II

Improving the approximation

```
interval real_rep::improve (double new_cost) {
  if (new_cost <= cost) return best;
  cost= max (new_cost, 2.0 * cost);
  set_precision (precision_for (cost));
  best= compute ();
  restore_precision ();
  return best;
}

interval real_rep::approximate (const dyadic& err) {
  while (radius (best) >= err)
    (void) improve (2 * cost);
  return best;
}
```

Model for complexity analysis

Global approximation problem

Input: an acyclic graph G with for each node $\alpha \in G$:

- A real function f_α from the library.

  Induces by induction a real number $x_\alpha = f_\alpha(x_{\alpha[1]}, \ldots, x_{\alpha[|\alpha|]})$.

- A tolerance $\varepsilon_\alpha \in \mathbb{D}^>$.

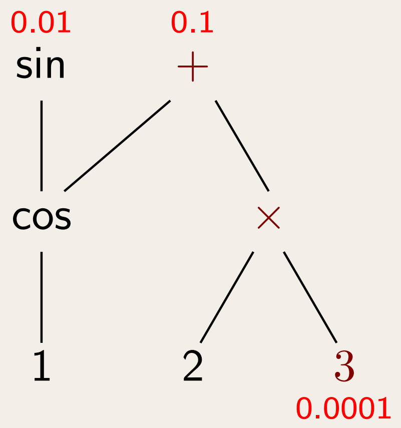
Output: for each node an interval $\boldsymbol{x}_\alpha \ni x_\alpha$ with

- $r_{\boldsymbol{x}_\alpha} < \varepsilon_\alpha$.
- $\boldsymbol{x}_\alpha \supseteq f_\alpha(\boldsymbol{x}_{\alpha[1]}, \ldots, \boldsymbol{x}_{\alpha[|\alpha|]})$.

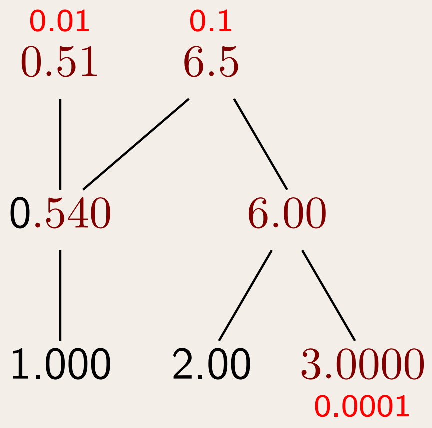
Drawbacks

- Does not model incremental computations.
- No dependency of computations on intermediate results.

Example



Example



Total versus final computation cost

For each node $\alpha \in G$, let

- $T_{\alpha,1}, \ldots, T_{\alpha,p_\alpha}$: requested costs of the successive evaluations of x_α.
- $t_{\alpha,1}, \ldots, t_{\alpha,p_\alpha}$: actual costs of the successive evaluations of x_α.
- $t_\alpha = t_{\alpha,1} + \cdots + t_{\alpha,p_\alpha}$ and $t_\alpha^{\text{fin}} = t_{\alpha,p_\alpha}$

By construction $T_{\alpha,1} = 1, T_{\alpha,2} = 2, T_{\alpha,3} = 4, \ldots$.

However, we do not necessarily have $t_{\alpha,i} = T_{\alpha,i}$, unless α is a leaf.

Indeed: $x = -y$, where y admits a slow approximation algorithm.

Total versus final computation cost

For each node $\alpha \in G$, let

- $T_{\alpha,1}, \ldots, T_{\alpha,p_\alpha}$: requested costs of the successive evaluations of x_α.
- $t_{\alpha,1}, \ldots, t_{\alpha,p_\alpha}$: actual costs of the successive evaluations of x_α.
- $t_\alpha = t_{\alpha,1} + \cdots + t_{\alpha,p_\alpha}$ and $t_\alpha^{\text{fin}} = t_{\alpha,p_\alpha}$

Nevertheless: for each α, we have $t_{\alpha,1} \leqslant \cdots \leqslant t_{\alpha,p_\alpha}$.

Let λ be a node for which $p = p_\lambda$ is maximal (necessarily a leaf).

Setting $t = \sum_\alpha t_\alpha$ and $t^{\text{fin}} = \sum_\alpha t_\alpha^{\text{fin}}$, we then have

$$t = \sum_{\alpha,i} t_{\alpha,i} \leqslant \sum_\alpha p_\alpha t_\alpha^{\text{fin}} \leqslant p\, t^{\text{fin}}.$$

It follows that

$$t^{\text{fin}} \leqslant t \leqslant (\log_2 t^{\text{fin}})\, t^{\text{fin}}.$$

Final versus optimal computation cost

Now consider an optimal solution and let

- t_α^{opt} time spent to compute $\boldsymbol{x}_\alpha$.
- $t^{\text{opt}} = \sum_\alpha t_\alpha^{\text{opt}}$.

Denoting by s the size of G, it can be shown that

$$t^{\text{opt}} \leqslant t^{\text{fin}} \leqslant 2\, s\, t^{\text{opt}}.$$

This bound is nevertheless "optimal":

$$y = x_1 \cdots x_n,$$

with $x_1 = \cdots = x_n = 0$ and exactly one of the x_i has a slow approximation algorithm.