Wykorzystanie biblioteki interval boost

Przemysław Spurek

18 sierpnia 2010

Problem:
Komputer przybliża liczby rzeczywiste np. $\sqrt{2}$.

Rozwiązanie:
Artmetyka interwałowa.

Źródło:

`http://www.boost.org`

Przykład zastosowania:
W sierpniu 1998 r. Tom Hales (hales@math. lsa.umich.edu) ogłosił dowód słynnej hipotezy Keplera, będącej na chyba jeszcze słynniejszej liście problemów matematycznych przedstawionej przez Davida Hilberta w 1900 r. Treścią hipotezy było przypuszczenie, że żaden układ identycznych kul nie może przewyższać gęstością struktury powierzchniowo centrowanej fcc. Wszyscy to wiedzieliśmy z kursu fizyki ciała stałego, jednakże ścisły dowód stał sie możliwy dopiero dzięki metodom interwałowym

Artmetyka interwałowa działa na odcinkach.

Definicja 1

Odcinek to domknięty i zwarty podzbiór zbioru $\mathbb{F}$ nazywanego base number type.
Gdzie $\mathbb{F}$ może być równe:

- $\mathbb{R}$
- podzbiór $\mathbb{R}$
- $\mathbb{N}$
- ...

Artmetyka interwałowa działa na odcinkach.

Definicja 1

Odcinek to domknięty i zwarty podzbiór zbioru $\mathbb{F}$ nazywanego base number type.
Gdzie $\mathbb{F}$ może być równe:

- $\mathbb{R}$
- podzbiór $\mathbb{R}$
- $\mathbb{N}$
- ...

Artmetyka interwałowa działa na odcinkach.

Definicja 1

Odcinek to domknięty i zwarty podzbiór zbioru $\mathbb{F}$ nazywanego base number type.
Gdzie $\mathbb{F}$ może być równe:

- $\mathbb{R}$
- podzbiór $\mathbb{R}$
- $\mathbb{N}$
- ...

Artmetyka interwałowa działa na odcinkach.

Definicja 1

Odcinek to domknięty i zwarty podzbiór zbioru $\mathbb{F}$ nazywanego base number type.
Gdzie $\mathbb{F}$ może być równe:

- $\mathbb{R}$
- podzbiór $\mathbb{R}$
- $\mathbb{N}$
- ...

Artmetyka interwałowa działa na odcinkach.

Definicja 1

Odcinek to domknięty i zwarty podzbiór zbioru $\mathbb{F}$ nazywanego base number type.
Gdzie $\mathbb{F}$ może być równe:

- $\mathbb{R}$
- podzbiór $\mathbb{R}$
- $\mathbb{N}$
- ...

Wyróżniamy trzy rodzaje odcinków:

- ograniczone

$$[a, b] \quad gdzie \quad a, b \in \mathbb{F} \quad oraz \quad a \leq b$$

- niegraniczone

$$[-\infty, a], [b, +\infty], [-\infty, +\infty]$$

- pusty

$$[a, b] \quad gdzie \quad a, b \in \mathbb{F} \quad oraz \quad a > b$$

Wyróżniamy trzy rodzaje odcinków:

- ograniczone

$$[a, b] \quad gdzie \quad a, b \in \mathbb{F} \quad oraz \quad a \leq b$$

- niegraniczone

$$[-\infty, a], [b, +\infty], [-\infty, +\infty]$$

- pusty

$$[a, b] \quad gdzie \quad a, b \in \mathbb{F} \quad oraz \quad a > b$$

Wyróżniamy trzy rodzaje odcinków:

- ograniczone

$$[a, b] \quad gdzie \quad a, b \in \mathbb{F} \quad oraz \quad a \leq b$$

- niegraniczone

$$[-\infty, a], [b, +\infty], [-\infty, +\infty]$$

- pusty

$$[a, b] \quad gdzie \quad a, b \in \mathbb{F} \quad oraz \quad a > b$$

Wyróżniamy trzy rodzaje odcinków:

- ograniczone

$$[a, b] \quad gdzie \quad a, b \in \mathbb{F} \quad oraz \quad a \leq b$$

- niegraniczone

$$[-\infty, a], [b, +\infty], [-\infty, +\infty]$$

- pusty

$$[a, b] \quad gdzie \quad a, b \in \mathbb{F} \quad oraz \quad a > b$$

Oznaczenie:
Dla odcinka $[x] = [\underline{x}, \overline{x}]$
$\underline{x}$ nazywamy lower bound
$\overline{x}$ nazywamy upper bound

```
interval<double> I1(2);
cout<< I1.lower() <<" , "<< I1.upper() <<endl;
```

```
2,2
```

Oznaczenie:
Dla odcinka $[x] = [\underline{x}, \overline{x}]$
$\underline{x}$ nazywamy lower bound
$\overline{x}$ nazywamy upper bound

```
interval<double> I1(2);
cout<< I1.lower() <<" , "<< I1.upper() <<endl;
```

2 , 2

Jeżeli nie da się wyznaczyć dokładnej wartości $\overline{x}$ lub $\underline{x}$ to musimy je zaokrąglić:
$\diamond[x] = [\bigtriangledown\underline{x}, \bigtriangleup\overline{x}]$
$\bigtriangledown\underline{x}$ nazywamy lower bound
$\bigtriangleup\overline{x}$ nazywamy upper bound

```
interval<double> I1(2);
interval<double> I2 = sqrt( I1 );
cout<< I2.lower() <<" , "<< I2.upper() <<endl;
```

```
1.41421 , 1.41421
```

```
cout<< I2.lower() - 1.41421<<endl;
```

```
3.56237e-006
```

Jeżeli nie da się wyznaczyć dokładnej wartości $\overline{x}$ lub $\underline{x}$ to musimy je zaokrąglić:
$\diamond[x] = [\bigtriangledown\underline{x}, \bigtriangleup\overline{x}]$
$\bigtriangledown\underline{x}$ nazywamy lower bound
$\bigtriangleup\overline{x}$ nazywamy upper bound

```
interval<double> I1(2);
interval<double> I2 = sqrt( I1 );
cout<< I2.lower() <<" , "<< I2.upper() <<endl;
```

1.41421 , 1.41421

```
cout<< I2.lower() - 1.41421<<endl;
```

3.56237e-006

Jeżeli nie da się wyznaczyć dokładnej wartości $\overline{x}$ lub $\underline{x}$ to musimy je zaokrąglić:
$\diamond[x] = [\bigtriangledown\underline{x}, \bigtriangleup\overline{x}]$
$\bigtriangledown\underline{x}$ nazywamy lower bound
$\bigtriangleup\overline{x}$ nazywamy upper bound

```
interval<double> I1(2);
interval<double> I2 = sqrt( I1 );
cout<< I2.lower() <<" , "<< I2.upper() <<endl;
```

1.41421 , 1.41421

```
cout<< I2.lower() - 1.41421<<endl;
```

3.56237e-006

Klasa template interval<T> zawiera dwa parametry:

- T
  Określa on zbiór $\mathbb{F}$
  - $\mathbb{R}$
  - podzbiór $\mathbb{R}$
  - $\mathbb{N}$
  - ...
- Policies
  Nie określa on typu zbioru $\mathbb{F}$ tylko zachowanie przedziałów.
  Zawiera on referencje do dwóch typów:
  - rounding policy
  - checking policy

Klasa template interval<T> zawiera dwa parametry:

- T
  Określa on zbiór $\mathbb{F}$
  - $\mathbb{R}$
  - podzbiór $\mathbb{R}$
  - $\mathbb{N}$
  - ...
- Policies
  Nie określa on typu zbioru $\mathbb{F}$ tylko zachowanie przedziałów.
  Zawiera on referencje do dwóch typów:
  - rounding policy
  - checking policy

Klasa template interval<T> zawiera dwa parametry:

- T
  Określa on zbiór $\mathbb{F}$
  - $\mathbb{R}$
  - podzbiór $\mathbb{R}$
  - $\mathbb{N}$
  - ...
- Policies
  Nie określa on typu zbioru $\mathbb{F}$ tylko zachowanie przedziałów.
  Zawiera on referencje do dwóch typów:
  - rounding policy
  - checking policy

Klasa template interval<T> zawiera dwa parametry:

- T
  Określa on zbiór $\mathbb{F}$
  - $\mathbb{R}$
  - podzbiór $\mathbb{R}$
  - $\mathbb{N}$
  - ...
- Policies
  Nie określa on typu zbioru $\mathbb{F}$ tylko zachowanie przedziałów.
  Zawiera on referencje do dwóch typów:
  - rounding policy
  - checking policy

Klasa template interval<T> zawiera dwa parametry:

- T
  Określa on zbiór $\mathbb{F}$
  - $\mathbb{R}$
  - podzbiór $\mathbb{R}$
  - $\mathbb{N}$
  - ...
- Policies
  Nie określa on typu zbioru $\mathbb{F}$ tylko zachowanie przedziałów.
  Zawiera on referencje do dwóch typów:
  - rounding policy
  - checking policy

Klasa template interval<T> zawiera dwa parametry:

- T
  Określa on zbiór $\mathbb{F}$
  - $\mathbb{R}$
  - podzbiór $\mathbb{R}$
  - $\mathbb{N}$
  - ...
- Policies
  Nie określa on typu zbioru $\mathbb{F}$ tylko zachowanie przedziałów.
  Zawiera on referencje do dwóch typów:
  - rounding policy
  - checking policy

Klasa template interval<T> zawiera dwa parametry:

- T
  Określa on zbiór $\mathbb{F}$
  - $\mathbb{R}$
  - podzbiór $\mathbb{R}$
  - $\mathbb{N}$
  - ...
- Policies
  Nie określa on typu zbioru $\mathbb{F}$ tylko zachowanie przedziałów.
  Zawiera on referencje do dwóch typów:
  - rounding policy
  - checking policy

Klasa template interval<T> zawiera dwa parametry:

- T
  Określa on zbiór $\mathbb{F}$
  - $\mathbb{R}$
  - podzbiór $\mathbb{R}$
  - $\mathbb{N}$
  - ...
- Policies
  Nie określa on typu zbioru $\mathbb{F}$ tylko zachowanie przedziałów.
  Zawiera on referencje do dwóch typów:
  - rounding policy
  - checking policy

Klasa template interval<T> zawiera dwa parametry:

- T
  Określa on zbiór $\mathbb{F}$
  - $\mathbb{R}$
  - podzbiór $\mathbb{R}$
  - $\mathbb{N}$
  - ...
- Policies
  Nie określa on typu zbioru $\mathbb{F}$ tylko zachowanie przedziałów.
  Zawiera on referencje do dwóch typów:
  - rounding policy
  - checking policy

Klasa template interval<T> zawiera dwa parametry:

- T
  Określa on zbiór $\mathbb{F}$
  - $\mathbb{R}$
  - podzbiór $\mathbb{R}$
  - $\mathbb{N}$
  - ...
- Policies
  Nie określa on typu zbioru $\mathbb{F}$ tylko zachowanie przedziałów.
  Zawiera on referencje do dwóch typów:
  - rounding policy
  - checking policy

The rounding policy

- Określa sposób zaokrąglenia końców przedziałów

```
typedef interval<double,policies
<save_state<rounded_transc_exact<double> >
, checking_no_empty<double> > > I;
I i1(2);
I i2 = sqrt( i1 );
cout<<i2.lower() <<" , " <<i2.upper()<<endl;
```

1.41421 , 1.41421

The rounding policy

- Określa sposób zaokrąglenia końców przedziałów

```
rounded_transc_std,
rounded_transc_opp,
rounded_transc_exact,
rounded_transc_dummy
```

The rounding policy

- Określa sposób zaokrąglenia końców przedziałów
- Wbudowane dla typów podstawowych np. double, float

`rounded_transc_exact`

The rounding policy

- Określa sposób zaokrąglenia końców przedziałów
- Wbudowane dla typów podstawowych np. double, float
- Dla własnych typów możesz sam zdefiniować

  `interval_lib::policies`

The checking policy Opisuje jak klasa Interval radzi sobie z sytuacjami wyjątkowymi

- przedział pusty

  ```
  interval<double> I1(2 , 0);
  ```

  blad

The checking policy Opisuje jak klasa Interval radzi sobie z sytuacjami wyjątkowymi

- przedział pusty

```
typedef interval<double,policies
<save_state<rounded_transc_std<double> >
, checking_no_empty<double> > > K;
K k1(2 , 0);
```

blad

The checking policy Opisuje jak klasa Interval radzi sobie z sytuacjami wyjątkowymi

- przedział pusty

```
typedef interval<double,policies
<save_state<rounded_transc_std<double> >
, checking_no_nan<double> > > K;
K k1(2 , 0);
cout<<k1.lower() <<" , " <<k1.upper()<<endl;
```

nan , nan

- dzielenie przez zero
- przedziały nieograniczone
- ...

The checking policy Opisuje jak klasa Interval radzi sobie z sytuacjami wyjątkowymi

- przedział pusty

```
typedef interval<double,policies
<save_state<rounded_transc_std<double> >
, checking_no_nan<double> > > K;
K k1(2 , 0);
cout<<k1.lower() <<" , " <<k1.upper()<<endl;
```

nan , nan

- dzielenie przez zero
- przedziały nieograniczone
- ...

The checking policy Opisuje jak klasa Interval radzi sobie z sytuacjami wyjątkowymi

- przedział pusty

```
typedef interval<double,policies
<save_state<rounded_transc_std<double> >
, checking_no_nan<double> > > K;
K k1(2 , 0);
cout<<k1.lower() <<" , " <<k1.upper()<<endl;
```

nan , nan

- dzielenie przez zero
- przedziały nieograniczone
- ...

The checking policy Opisuje jak klasa Interval radzi sobie z sytuacjami wyjątkowymi

- przedział pusty

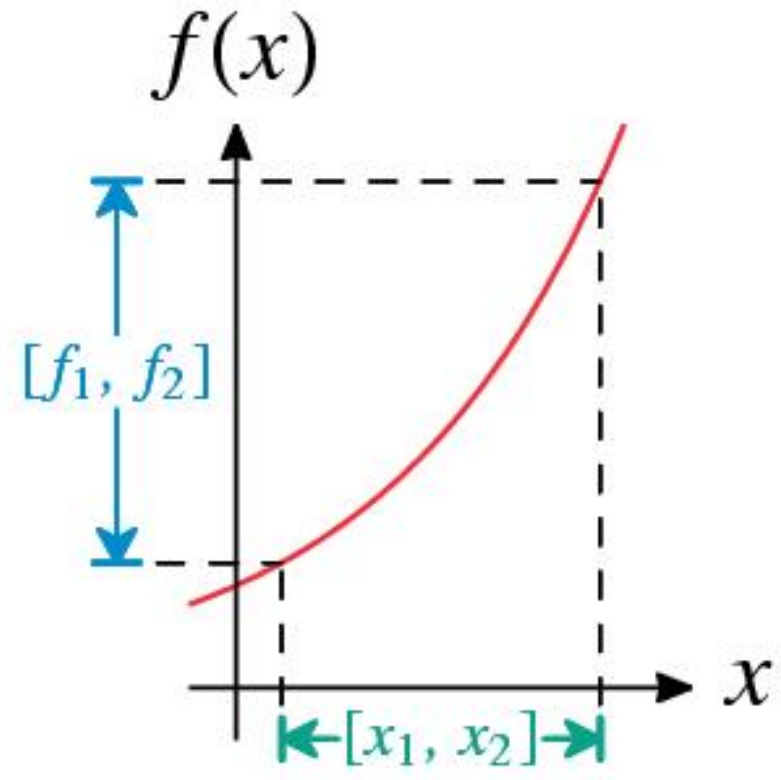
```
typedef interval<double,policies
<save_state<rounded_transc_std<double> >
, checking_no_nan<double> > > K;
K k1(2 , 0);
cout<<k1.lower() <<" , " <<k1.upper()<<endl;
```

nan , nan

- dzielenie przez zero
- przedziały nieograniczone
- ...

Definicja 2

$$f([x]) = [f(x)] = \{f(x) : x \in [x]\}$$



Definicja 3

$$f([x]) = [f(x)] = \{f(x) : x \in [x]\}$$

Podstawowe operacje:
Niech $[x] = [\underline{x}, \overline{x}]$ oraz $[y] = [\underline{y}, \overline{y}]$

- $[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$

```
interval<double> I1( 0 , 1 );
interval<double> I2( 0 , 1 );
interval<double> I3 = I1 + I2;
cout<< I3.lower() <<" , "<< I3.upper() <<endl;
```

```
0 , 2
```

Definicja 3

$$f([x]) = [f(x)] = \{f(x) : x \in [x]\}$$

Podstawowe operacje:
Niech $[x] = [\underline{x}, \overline{x}]$ oraz $[y] = [\underline{y}, \overline{y}]$

- $[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$

```
interval<double> I1( 0 , 1 );
interval<double> I2( 0 , 1 );
interval<double> I3 = I1 + I2;
cout<< I3.lower() <<" , "<< I3.upper() <<endl;
```

0 , 2

Definicja 4

$$f([x]) = [f(x)] = \{f(x) : x \in [x]\}$$

Podstawowe operacje:
Niech $[x] = [\underline{x}, \overline{x}]$ oraz $[y] = [\underline{y}, \overline{y}]$

- $[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[x] - [y] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$

```
interval<double> I1( 0 , 1 );
interval<double> I2( 0 , 1 );
interval<double> I3 = I1 - I2;
cout<< I3.lower() <<" , "<< I3.upper() <<endl;
```

-1 , 1

Definicja 5

$$f([x]) = [f(x)] = \{f(x) : x \in [x]\}$$

Podstawowe operacje:
Niech $[x] = [\underline{x}, \overline{x}]$ oraz $[y] = [\underline{y}, \overline{y}]$

- $[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[x] - [y] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$
- $[x] * [y] = [\min\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}]$

```
interval<double> I1( 0 , 1 );
interval<double> I2( 0 , 1 );
interval<double> I3 = I1 * I2;
cout<< I3.lower() <<" , "<< I3.upper() <<endl;
```

0 , 1

Definicja 6

$$f([x]) = [f(x)] = \{f(x) : x \in [x]\}$$

Podstawowe operacje:
Niech $[x] = [\underline{x}, \overline{x}]$ oraz $[y] = [\underline{y}, \overline{y}]$

- $[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[x] - [y] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$
- $[x] * [y] = [\min\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}]$

```
interval<double> I1( 0.2 , 4 );
interval<double> I2( 1 , 2 );
interval<double> I3 = I1 * I2;
cout<< I3.lower() <<" , "<< I3.upper() <<endl;
```

```
0.2 , 8
```

Definicja 6

$$f([x]) = [f(x)] = \{f(x) : x \in [x]\}$$

Podstawowe operacje:
Niech $[x] = [\underline{x}, \overline{x}]$ oraz $[y] = [\underline{y}, \overline{y}]$

- $[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[x] - [y] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$
- $[x] * [y] = [\min\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}]$

```
interval<double> I1( 0.2 , 4 );
interval<double> I2( 1 , 2 );
interval<double> I3 = I1 * I2;
cout<< I3.lower() <<" , "<< I3.upper() <<endl;
```

0.2 , 8

Definicja 7

$$f([x]) = [f(x)] = \{f(x) : x \in [x]\}$$

Podstawowe operacje:
Niech $[x] = [\underline{x}, \overline{x}]$ oraz $[y] = [\underline{y}, \overline{y}]$

- $[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[x] - [y] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$
- $[x] * [y] = [\min\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}]$

```
typedef interval<double,policies
<save_state<rounded_transc_std<double> >
, checking_no_nan<double> > > K;
K k1(2 , 0);
K k2(1 , 3);
K k3 = k1 * k2;
cout<<k3.lower() <<" , " <<k3.upper()<<endl;
```

```
nan , nan
```

Definicja 7

$$f([x]) = [f(x)] = \{f(x) : x \in [x]\}$$

Podstawowe operacje:
Niech $[x] = [\underline{x}, \overline{x}]$ oraz $[y] = [\underline{y}, \overline{y}]$

- $[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[x] - [y] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$
- $[x] * [y] = [\min\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}]$

```
typedef interval<double,policies
<save_state<rounded_transc_std<double> >
, checking_no_nan<double> > > K;
K k1(2 , 0);
K k2(1 , 3);
K k3 = k1 * k2;
cout<<k3.lower() <<" , " <<k3.upper()<<endl;
```

```
nan , nan
```

Definicja 8

$$f([x]) = [f(x)] = \{f(x) : x \in [x]\}$$

Podstawowe operacje:
Niech $[x] = [\underline{x}, \overline{x}]$ oraz $[y] = [\underline{y}, \overline{y}]$

- $[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[x] - [y] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$
- $[x] * [y] = [\min\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}]$
- $[x] \div [y] = [x][\frac{1}{\overline{y}}, \frac{1}{\underline{y}}]$ gdzie $0 \in [y]$

```
interval<double> I1( 2 , 4 );
interval<double> I2( 1 , 2 );
interval<double> I3 = I1 / I2;
cout<< I3.lower() <<" , "<< I3.upper() <<endl;
```

1 , 4

Definicja 9

$$f([x]) = [f(x)] = \{f(x) : x \in [x]\}$$

Podstawowe operacje:
Niech $[x] = [\underline{x}, \overline{x}]$ oraz $[y] = [\underline{y}, \overline{y}]$

- $[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
- $[x] - [y] = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$
- $[x] * [y] = [\min\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}]$
- $[x] \div [y] = [x][\frac{1}{\overline{y}}, \frac{1}{\underline{y}}]$ gdzie $0 \in [y]$
- anlogicznie zdefiniowane są działania: sin, cos , pow , sqrt ...

```
interval<double> I1( 2 , 4 );
interval<double> I2 = pow( I1 , 2 );
cout<< I2.lower() <<" , "<< I2.upper() <<endl;
```

4 , 16

Problemy:

- Dzielenie przez zero:
  $\frac{1}{[-1,1]} = [-\infty, -1] \cup [1, +\infty]$

```
interval<double> I1( 1 );
interval<double> I2( -1 , 1 );
interval<double> I3 = I1 / I2;
cout<< I3.lower() <<" , "<< I3.upper() <<endl;
cout<<zero_in(I3)<<endl;
```

  -inf , inf
  1
- Dzielenie przez nieskończoność:
  $\frac{1}{[1,+\infty]} = [0, 1] \setminus \{0\}$
- Podobne problemy z funkcjami kawałkami ciągłymi:
  $\tan[\frac{\pi}{4}, \frac{3\pi}{4}] = [-\infty, -1] \cup [1, +\infty]$

Problemy:

- Dzielenie przez zero:
  $\frac{1}{[-1,1]} = [-\infty, -1] \cup [1, +\infty]$

```
interval<double> I1( 1 );
interval<double> I2( -1 , 1 );
interval<double> I3 = I1 / I2;
cout<< I3.lower() <<" , "<< I3.upper() <<endl;
cout<<zero_in(I3)<<endl;
```

  -inf , inf
  1
- Dzielenie przez nieskończoność:
  $\frac{1}{[1,+\infty]} = [0, 1] \setminus \{0\}$
- Podobne problemy z funkcjami kawałkami ciągłymi:
  $\tan[\frac{\pi}{4}, \frac{3\pi}{4}] = [-\infty, -1] \cup [1, +\infty]$

Problemy:

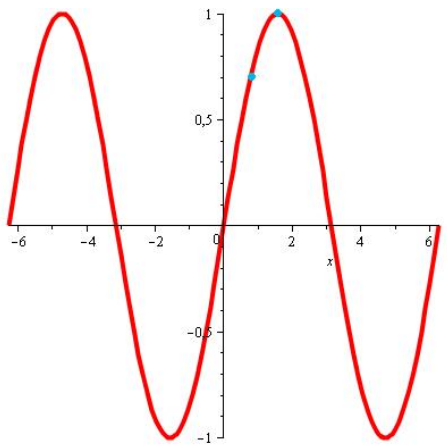
- Dzielenie przez zero:
  $\frac{1}{[-1,1]} = [-\infty, -1] \cup [1, +\infty]$

```
interval<double> I1( 1 );
interval<double> I2( -1 , 1 );
interval<double> I3 = I1 / I2;
cout<< I3.lower() <<" , "<< I3.upper() <<endl;
cout<<zero_in(I3)<<endl;
```

  -inf , inf
  1
- Dzielenie przez nieskończoność:
  $\frac{1}{[1,+\infty]} = [0, 1] \setminus \{0\}$
- Podobne problemy z funkcjami kawałkami ciągłymi:
  $\tan[\frac{\pi}{4}, \frac{3\pi}{4}] = [-\infty, -1] \cup [1, +\infty]$

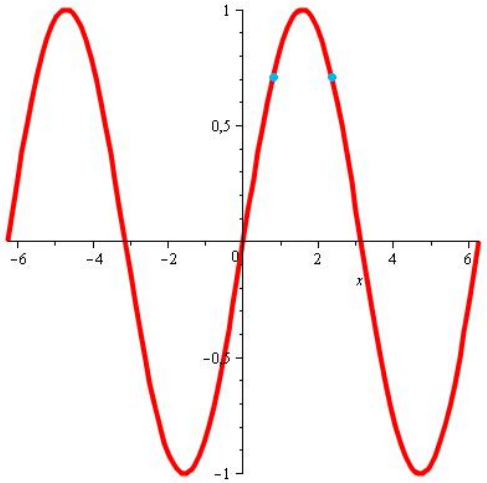
```
typedef interval<double,policies
<save_state<rounded_transc_exact<double> >
, checking_no_nan<double> > > K;
K k1( M_PI/4 , M_PI/2);
K k2= sin(k1);
cout<<k2.lower() <<" , " <<k2.upper()<<endl;
```

0.707107 , 1



```
typedef interval<double,policies
<save_state<rounded_transc_exact<double> >
, checking_no_nan<double> > > K;
K l1( M_PI/4 , (3*M_PI)/4);
K l2= sin(l1);
cout<<l2.lower() <<" , " <<l2.upper()<<endl;
cout<<in( 0.8 , l2)<<endl;
```

0.707107 , 1
1



Definicja 10

Operatory porownania:

$$\mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{B}$$

Gdzie $\mathbb{B}$ to boolean set.

$\mathbb{B} = \{\emptyset, \{true\}, \{false\}, \{true, false\}\}$

Przykład:

```
interval<double> I1( 0 , 2 );
interval<double> I2( 3 , 4 );
if( I1 < I2 ){
cout<<" I1 < I2 "<<endl;
}
```

I1 < I2

Definicja 10

Operatory porownania:

$$\mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{B}$$

Gdzie $\mathbb{B}$ to boolean set.
$\mathbb{B} = \{\emptyset, \{true\}, \{false\}, \{true, false\}\}$

Przykład:

```
interval<double> I1( 0 , 2 );
interval<double> I2( 3 , 4 );
if( I1 < I2 ){
cout<<" I1 < I2 "<<endl;
}
```

I1 < I2

Definicja 10

Operatory porownania:

$$\mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{B}$$

Gdzie $\mathbb{B}$ to boolean set.
$\mathbb{B} = \{\emptyset, \{\mathit{true}\}, \{\mathit{false}\}, \{\mathit{true}, \mathit{false}\}\}$

Przykład:

```
interval<double> I1( 0 , 2 );
interval<double> I2( 3 , 4 );
if( I1 < I2 ){
cout<<" I1 < I2 "<<endl;
}
```

I1 < I2

Problemy:

- Jeden element jest pusty
  Umiemy rozwiązać
- $[\underline{x}, \overline{x}] < [\underline{y}, \overline{y}]$ zwróci $\{true, false\}$ gdy
  $\exists \quad x_1 x_2 \in [\underline{x}, \overline{x}]$ oraz $\exists \quad y_1 y_2 \in [\underline{y}, \overline{y}]$ $x_1 \geq y_1$ i $x_2 < y_2$

```
interval<double> I1( 1 , 3 );
interval<double> I2( 2 , 4 );
if( I1 < I2 ){
cout<<" I1 < I2 "<<endl;
}
```

  blad

Problemy:

- Jeden element jest pusty
  Umiemy rozwiązać
- $[\underline{x}, \overline{x}] < [\underline{y}, \overline{y}]$ zwróci $\{true, false\}$ gdy
  $\exists \quad x_1 x_2 \in [\underline{x}, \overline{x}]$ oraz $\exists \quad y_1 y_2 \in [\underline{y}, \overline{y}]$ $x_1 \geq y_1$ i $x_2 < y_2$

```
interval<double> I1( 1 , 3 );
interval<double> I2( 2 , 4 );
if( I1 < I2 ){
cout<<" I1 < I2 "<<endl;
}
```

blad

Problemy:

- Jeden element jest pusty
  Umiemy rozwiązać
- $[\underline{x}, \overline{x}] < [\underline{y}, \overline{y}]$ zwróci $\{true, false\}$ gdy
  $\exists \quad x_1 x_2 \in [\underline{x}, \overline{x}]$ oraz $\exists \quad y_1 y_2 \in [\underline{y}, \overline{y}]$ $x_1 \geq y_1$ i $x_2 < y_2$

```
interval<double> I1( 1 , 3 );
interval<double> I2( 2 , 4 );
if( I1 < I2 ){
cout<<" I1 < I2 "<<endl;
}
```

blad

Problemy:

- Jeden element jest pusty
  Umiemy rozwiązać
- $[\underline{x}, \overline{x}] < [\underline{y}, \overline{y}]$ zwróci $\{true, false\}$ gdy
  $\exists \quad x_1 x_2 \in [\underline{x}, \overline{x}]$ oraz $\exists \quad y_1 y_2 \in [\underline{y}, \overline{y}]$ $x_1 \geq y_1$ i $x_2 < y_2$

```
using namespace compare::certain;
interval<double> I1( 1 , 3 );
interval<double> I2( 2 , 4 );
if( I1 < I2 ){
cout<<" I1 < I2 "<<endl;
}
```

Problemy:

- Jeden element jest pusty
  Umiemy rozwiązać
- $[\underline{x}, \overline{x}] < [\underline{y}, \overline{y}]$ zwróci $\{true, false\}$ gdy
  $\exists \quad x_1 x_2 \in [\underline{x}, \overline{x}]$ oraz $\exists \quad y_1 y_2 \in [\underline{y}, \overline{y}]$ $x_1 \geq y_1$ i $x_2 < y_2$

```
using namespace compare::lexicographic
interval<double> I1( 1 , 3 );
interval<double> I2( 2 , 4 );
if( I1 < I2 ){
cout<<" I1 < I2 "<<endl;
}
```

I1 < I2

KONIEC