Простейшая реализация интервальной арифметики

В.А.Шишкин

6 июня 2015 г.

Содержание

1 Интервальная арифметика

Арифметические операции над интервалами $x = [x_{\min}, x_{\max}]$ и $y = [y_{\min}, y_{\max}]$ задаются следующими выражениями [1, стр. 16]:

$$
\begin{aligned}
x + y &= [x_{\min} + y_{\min}, x_{\max} + y_{\max}] \\
x - y &= [x_{\min} - y_{\max}, x_{\max} - y_{\min}] \\
x \cdot y &= \big[\min(x_{\min}y_{\min}, x_{\min}y_{\max}, x_{\max}y_{\min}, x_{\max}y_{\max}), \\
&\qquad \max(x_{\min}y_{\min}, x_{\min}y_{\max}, x_{\max}y_{\min}, x_{\max}y_{\max})\big] \\
x/y &= x \cdot [1/y_{\max}, 1/y_{\min}]
\end{aligned}
$$

2 Реализация

2.1 Электронные таблицы

2.1.1 Microsoft Excel

```
Rem x + y = [x_min + y_min, x_max + y_max]
Function int_add(x, y)
  Dim res(2) As Double
  res(0) = x(1) + y(1): res(1) = x(2) + y(2)
  int_add = res
End Function

Rem x - y = [x_min - y_max, x_max - y_min]
Function int_subt(x, y)
  Dim res(2) As Double
  res(0) = x(1) - y(2):  res(1) = x(2) - y(1)
  int_subt = res
End Function

Function min(x, y)
    If x < y Then min = x Else min = y
End Function
Function max(x, y)
    If x > y Then max = x Else max = y
End Function

Rem x · y = [min(x_min y_min, x_min y_max, x_max y_min, x_max y_max),
Rem          max(x_min y_min, x_min y_max, x_max y_min, x_max y_max)]
Function int_mul(x, y)
  Dim tmp(4) As Double
  Dim res(2) As Double

  tmp(0) = x(1) * y(1):  tmp(1) = x(1) * y(2)
  tmp(2) = x(2) * y(1):  tmp(3) = x(2) * y(2)

  res(0) = min(min(tmp(0), tmp(1)), min(tmp(2), tmp(3)))
  res(1) = max(max(tmp(0), tmp(1)), max(tmp(2), tmp(3)))

  int_mul = res
End Function

Rem x/y = x · [1/y_max, 1/y_min]
Function int_div(x, y)
  Dim tmp(2) As Double
  tmp(1) = 1 / y(2):  tmp(2) = 1 / y(1)
  int_div = int_mul(x, tmp)
End Function

Rem ℝ → 𝕀ℝ
Function interval(x As Double)
    Dim res(2) As Double
    res(0) = x: res(1) = x
    interval = res
End Function
```

2.1.2 Calc

Реализация в Calc почти аналогична реализации в Excel, за исключением различия в индексации диапазонов-массивов:

```
function int_add (x,y)
  dim res(1) as double
  res(0) = x(1,1)+y(1,1):  res(1) = x(1,2)+y(1,2)
  int_add = res
end function

function int_subt (x,y)
  dim res(1) as double
  res(0) = x(1,1)-y(1,2):  res(1) = x(1,2)-y(1,1)
  int_subt = res
end function

function int_mul (x,y)
  dim tmp(4) as double
  dim res(1) as double
  dim i as integer

  tmp(0) = x(1,1)*y(1,1):  tmp(1) = x(1,1)*y(1,2)
  tmp(2) = x(1,2)*y(1,1):  tmp(3) = x(1,2)*y(1,2)

  res(0) = tmp(0)
  res(1) = tmp(1)
  for i=1 to 3
    if tmp(i) < res(0) then res(0) = tmp(i)
    if tmp(i) > res(1) then res(1) = tmp(i)
  next i

  int_mul = res
end function

function int_div (x,y)
  dim tmp(4) as double
  dim res(1) as double
  dim i as integer

  tmp(0) = x(1,1)/y(1,1):  tmp(1) = x(1,1)/y(1,2)
  tmp(2) = x(1,2)/y(1,1):  tmp(3) = x(1,2)/y(1,2)

  res(0) = tmp(0)
  res(1) = tmp(1)
  for i=1 to 3
    if tmp(i) < res(0) then res(0) = tmp(i)
    if tmp(i) > res(1) then res(1) = tmp(i)
  next i

  int_div = res
end function

Function interval(x As Double)
    Dim res(2) As Double
    res(0) = x: res(1) = x
```

```
    interval = res
End Function
```

2.2 GNU Octave

Каждая процедура записывается в отдельный `m`-файл, название которого совпадает с названием процедуры. Все файлы хранятся в подкаталоге `@interval`.

```
## -*- texinfo -*-
## @deftypefn Function File  interval ()
## @deftypefnx Function File  interval (@var{a})
## @deftypefnx Function File  interval (@var{a1},@var{a2})
## @deftypefnx Function File  interval ([@var{a1},@var{a2}])
## Creates an interval object representing the interval number
##
## @example
## [a1,a2]
## @end example
##
## from a number 'a', interval [a1,a2] or two numbers (a1,a2)
## @end deftypefn
function x = interval (varargin)
  switch (nargin)
  # нет аргументов — интервал [0,0]
  case 0
    x.lh = [0,0];
    x = class(x,'interval');
  # один аргумент — результат зависит от типа
  case 1
    # интервал
    if (strcmp(class(varargin{1}),'interval'))
      x = a;
    # число — результат [a,a]
    elseif (isreal(varargin{1}) && length(varargin{1})==1)
      x.lh = [varargin{1},varargin{1}];
      x = class(x,'interval');
    # вектор из двух элементов — результат [a1,a2]
    elseif (isvector(varargin{1}) && ...
            isreal(varargin{1}) && ...
            length(varargin{1})==2 && ...
            varargin{1}(1)<=varargin{1}(2))
      x.lh = varargin{1};
      x = class(x,'interval');
    else
      error(strcat('interval: expecting a real number or',
                   ' a vector [a1,a2], where is a1<=a2'));
    endif
  # два аргумента, числа — результат [a1,a2]
  case 2
    if (isreal(varargin{1}) && length(varargin{1})==1 && ...
        isreal(varargin{2}) && length(varargin{2})==1 && ...
        varargin{1}<=varargin{2})
      x.lh = [varargin{1},varargin{2}];
      x = class(x,'interval');
    else
```

```
      error(strcat('interval: expecting two real numbers,',
                   ' a1 and a2, where is a1<=a2'));
    endif
  otherwise
    print_usage();
  endswitch
endfunction
```

Вывод информации об интервале в стандартный поток:

```
function display (x)
  if (x.lh(1) == x.lh(2))
    fprintf('%s = %f\n',inputname(1),x.lh(1));
  else
    fprintf('%s = [%f,%f]\n',inputname(1),x.lh(1),x.lh(2));
  endif
endfunction
```

Арифметические операции над интервалами:

```
# Унарный минус: −x
function res = uminus (x)
  res = interval(-x.lh(2),-x.lh(1));
endfunction

# Интервальное сложение: x + y
function res = plus (x,y)
  if (!strcmp(class(x),'interval')) x = interval(x); endif
  if (!strcmp(class(y),'interval')) y = interval(y); endif

  res = interval(x.lh + y.lh);
endfunction

# Интервальное вычитание: x − y
function res = minus (x,y)
  if (!strcmp(class(x),'interval')) x = interval(x); endif
  if (!strcmp(class(y),'interval')) y = interval(y); endif

  l = x.lh(1) - y.lh(2);
  h = x.lh(2) - y.lh(1);

  res = interval(l,h);
endfunction

# Интервальное умножение: x × y
function res = mtimes (x,y)
  if (!strcmp(class(x),'interval')) x = interval(x); endif
  if (!strcmp(class(y),'interval')) y = interval(y); endif

  b1 = x.lh(1) * y.lh(1);
  b2 = x.lh(1) * y.lh(2);
  b3 = x.lh(2) * y.lh(1);
  b4 = x.lh(2) * y.lh(2);

  res = interval(min([b1,b2,b3,b4]), ...
                 max([b1,b2,b3,b4]));
endfunction
```

```
# Интервальное деление: x/y, 0 ∉ y
function res = mrdivide (x,y)
  if (!strcmp(class(x),'interval')) x = interval(x); endif
  if (!strcmp(class(y),'interval')) y = interval(y); endif

  if (y.lh(1) <= 0 && 0 <= y.lh(2))
    error('mrdivide: zero into divider');
  endif

  res = x * interval(1/y.lh(2),1/y.lh(1));
endfunction
```

2.3 C++

2.3.1 Простейшая реализация

В заголовочном файле в пространстве имён `MATH` определяется класс `interval_t`:

```
#pragma once
#include <ostream>

namespace MATH{

  class interval_t
  {
    // Левая граница
    double _l;
    // Правая граница
    double _h;
  public:
    interval_t() : _l(0.0), _h(0.0) {}
    explicit interval_t(double x) : _l(x), _h(x) {}
    interval_t(double l, double h) : _l(l), _h(h)
    {
      if (_h < _l) throw "(interval_t) hi < lo";
    }
    interval_t(const interval_t& x) : _l(x._l), _h(x._h) {}

    interval_t& operator=(double x)
    {
      _l = _h = x;
      return *this;
    }
    interval_t& operator=(const interval_t& x)
    {
      _l = x._l;
      _h = x._h;
      return *this;
    }

    double lo() const { return _l; }
    double hi() const { return _h; }
  };

  // Унарный минус
```

```
  inline interval_t operator-(const interval_t& x)
  {
    return interval_t(-x.hi(), -x.lo());
  }
  // Сложение
  inline interval_t operator+(const interval_t& x, const interval_t& y)
  {
    return interval_t(x.lo() + y.lo(), x.hi() + y.hi());
  }
  // Вычитание
  inline interval_t operator-(const interval_t& x, const interval_t& y)
  {
    return interval_t(x.lo() - y.hi(), x.hi() - y.lo());
  }
  // Умножение
  interval_t operator*(const interval_t& x, const interval_t& y);
  // Деление
  interval_t operator/(const interval_t& x, const interval_t& y);

  std::ostream& operator<<(std::ostream& os, const interval_t& x);

}
```

Реализацию более сложных процедур выносим в отдельный cpp-файл.

```
namespace MATH {

  interval_t operator*(const interval_t& x, const interval_t& y)
  {
    auto t1 = x.lo()*y.lo();
    auto t2 = x.lo()*y.hi();
    auto t3 = x.hi()*y.lo();
    auto t4 = x.hi()*y.hi();

    return interval_t(__min(__min(t1, t2), __min(t3, t4)),
                      __max(__max(t1, t2), __max(t3, t4)));
  }
  interval_t operator/(const interval_t& x, const interval_t& y)
  {
    if (y.lo() <= 0.0 && 0.0 <= y.hi())
      throw "(interval_t) division by zero";

    return x*interval_t(1.0 / y.hi(), 1.0 / y.lo());
  }

  std::ostream& operator<<(std::ostream& os, const interval_t& x)
  {
    if (x.lo() == x.hi())
      os << x.lo();
    else
      os << "[" << x.lo() << ";" << x.hi() << "]";
    return os;
  }

}
```

2.3.2 Реализация с использованием направленного округления

В заголовочном файле в пространстве имён `MATH` определяется класс `interval_t`:

```
#ifndef __INTERVAL__
#define __INTERVAL__

// Depend on OS we will use _control87 (float.h, Windows)
// or fesetround (fenv.h, Linux)
#ifdef _WIN32
  #include <float.h>
#else
  #include <fenv.h>
#endif

// Only for std::min and std::max
#include <algorithm>
// Output into standard stream
#include <iostream>

namespace MATH
{
// Set rounding direction. To nearest number by default
#ifdef _WIN32
  inline
    void __set_round_down() { _control87(_RC_DOWN, _MCW_RC); }
  inline
    void __set_round_up() { _control87(_RC_UP, _MCW_RC); }
  inline
    void __set_round_default() { _control87(_CW_DEFAULT, _MCW_RC); }
#else
  inline
    void __set_round_down() { fesetround(FE_DOWNWARD); }
  inline
    void __set_round_up() { fesetround(FE_UPWARD); }
  inline
    void __set_round_default() { fesetround(FE_TONEAREST); }
#endif

  template <class T = double>
  class interval_t
  {
  public:
    // by default — exact zero: [0, 0]
    interval_t<T>() : _lb(0), _ub(0) {}
    // exact number [x, x]
    interval_t<T>(long number) : _lb(number), _ub(number) {}
    // number [x, x̄]
    interval_t<T>(long numer, long denom);
    // interval [x, ȳ]
    interval_t<T>(long numer_lb, long denom_lb,
                  long numer_ub, long denom_ub);

    T lo() const { return _lb; } // low bound
    T hi() const { return _ub; } // high bound

    // unary minus (negative number)
```

```
  interval_t<T> operator-()
  {
    interval_t<T> neg;
    neg._lb = -_ub;
    neg._ub = -_lb;
    return neg;
  }

  interval_t<T>& operator+=(const interval_t<T>& x);
  interval_t<T>& operator-=(const interval_t<T>& x);
  interval_t<T>& operator*=(const interval_t<T>& x);
  interval_t<T>& operator/=(const interval_t<T>& x);

  interval_t<T> operator + (const interval_t<T>& x)
  {
    return interval_t<T>(*this) += x;
  }
  interval_t<T> operator - (const interval_t<T>& x)
  {
    return interval_t<T>(*this) -= x;
  }
  interval_t<T> operator * (const interval_t<T>& x)
  {
    return interval_t<T>(*this) *= x;
  }
  interval_t<T> operator / (const interval_t<T>& x)
  {
    return interval_t<T>(*this) /= x;
  }

private:
  T _lb, _ub;
};
```

При определении границ интервала используются вычисления с направленным округлением

```
template <class T>
interval_t<T>::interval_t(long numer, long denom)
{
  __set_round_down();
  _lb = T(numer) / T(denom);
  __set_round_up();
  _ub = T(numer) / T(denom);
  __set_round_default();
}
template <class T>
interval_t<T>::interval_t(long numer_lb, long denom_lb,
                          long numer_ub, long denom_ub)
{
  __set_round_down();
  _lb = T(numer_lb) / T(denom_lb);
  __set_round_up();
  _ub = T(numer_ub) / T(denom_ub);
  __set_round_default();
}
```

Также направленное округление применяется при арифметических вычислениях:

```
template <class T>
interval_t<T>& interval_t<T>::operator += (const interval_t<T>& x)
{
  __set_round_down();
  _lb += x._lb;
  __set_round_up();
  _ub += x._ub;
  __set_round_default();

  return *this;
}
template <class T>
interval_t<T>& interval_t<T>::operator -= (const interval_t<T>& x)
{
  if (this == &x)
  {
    _lb = _ub = 0.0;
  }
  else
  {
    __set_round_down();
    long double l = _lb - x._ub;
    __set_round_up();
    long double u = _ub - x._lb;
    __set_round_default();

    _lb = l;
    _ub = u;
  }

  return *this;
}
template <class T>
interval_t<T>& interval_t<T>::operator *= (const interval_t<T>& x)
{
  if (this == &x) // x is same number
  {
    if (_lb > T(0) || _ub < T(0))   // _lb and _ub have same signs
    {
      __set_round_down();
      T l = std::min(_lb*_lb, _ub*_ub);
      __set_round_up();
      T u = std::max(_lb*_lb, _ub*_ub);
      __set_round_default();
      _lb = l;
      _ub = u;
    }
    else // _lb⩽0⩽_ub
    {
      _lb = T(0);
      __set_round_up();
      _ub = std::max(_lb*_lb, _ub*_ub);
      __set_round_default();
    }
```

```
    }
    else // x is different number
    {
      __set_round_down();
      T l = std::min(std::min(_lb*x._lb, _lb*x._ub),
                     std::min(_ub*x._lb, _ub*x._ub));
      __set_round_up();
      T u = std::max(std::max(_lb*x._lb, _lb*x._ub),
                     std::max(_ub*x._lb, _ub*x._ub));
      __set_round_default();

      _lb = l;
      _ub = u;
    }

    return *this;
  }
  template <class T>
  interval_t<T>& interval_t<T>::operator /= (const interval_t<T>& x)
  {
    if ((x._lb <= 0) && (0 <= x._ub)) throw "Division by zero";

    if (this == &x)
    {
      _lb = _ub = 1.0;
      return *this;
    }
    else
    {
      interval_t<T> x_inv;

      __set_round_down();
      x_inv._lb = T(1) / x._ub;
      __set_round_up();
      x_inv._ub = T(1) / x._lb;
      __set_round_default();

      return (*this) *= x_inv;
    }
  }
```

Вывод интервала в поток:

```
  template <class T>
  std::ostream& operator<< (std::ostream& os, interval_t<T>& x)
  {
    if (x.lo() == x.hi())
      os << x.lo();
    else
      os << "[" << x.lo() << ";" << x.hi() << "]";
    return os;
  }

} // Конец пространства имён MATH

#endif
```

2.4 CLIPS

```
; class of temporary objects
(defclass temporary
  (is-a USER)
  (role abstract)

  ; T — temporary object (will be deleted during garbage cleaning)
  ; F — permanent object
  (slot temporary (type SYMBOL)
                  (allowed-symbols T F) (default T))

  (message-handler delete-tmps)
)
(defmessage-handler temporary delete-tmps()
  (if (eq ?self:temporary T) then (send ?self delete))
)

; remove all temporary object from memory
(deffunction clear-temporary-garbage()
  (do-for-all-instances ((?x temporary))
                        (send ?x delete-tmps))
  (return)
)
```

```
(defclass interval
  (is-a temporary)
  (role concrete)

  (slot a (type NUMBER)) ; low bound
  (slot b (type NUMBER)) ; high bound

  (message-handler show)
)
; print interval
(defmessage-handler interval show ()
  (printout t "[" ?self:a ";" ?self:b "]")
)

; create interval approximation of the number
(defmethod create-interval ((?x NUMBER))
  (make-instance of interval (a ?x) (b ?x))
)
; create interval
(defmethod create-interval ((?xl NUMBER) (?xt NUMBER))
  (make-instance of interval (a ?xl) (b ?xt))
)

; [x1, x2] + [y1, y2] = [x1 + y1, x2 + y2]
(defmethod + ((?x interval) (?y interval))
  (make-instance of interval
    (a (+ (send ?x get-a) (send ?y get-a)))
    (b (+ (send ?x get-b) (send ?y get-b)))
  )
)
; [x1, x2] − [y1, y2] = [x1 − y2, x2 − y1]
```

```
(defmethod - ((?x interval) (?y interval))
  (make-instance of interval
    (a (- (send ?x get-a) (send ?y get-b)))
    (b (- (send ?x get-b) (send ?y get-a)))
  )
)
; [x1, x2] × [y1, y2] = [min(x1y1, x1y2, x2y1, x2y2),
;                        max(x1y1, x1y2, x2y1, x2y2)]
(defmethod * ((?x interval) (?y interval))
  (bind ?x1 (* (send ?x get-a) (send ?y get-a)))
  (bind ?x2 (* (send ?x get-a) (send ?y get-b)))
  (bind ?x3 (* (send ?x get-b) (send ?y get-a)))
  (bind ?x4 (* (send ?x get-b) (send ?y get-b)))

  (make-instance of interval
    (a (min ?x1 ?x2 ?x3 ?x4))
    (b (max ?x1 ?x2 ?x3 ?x4))
  )
)
; [x1, x2]/[y1, y2] = [x1, x2] × [1/y2, 1/y1], 0 ∉ [y1, y2]
(defmethod / ((?x interval) (?y interval))
  (if (and (<= (send ?y get-a) 0)
           (>= (send ?y get-b) 0))
  then
    (printout t "Division by zero!" crlf)
    (halt)
  )

  (bind ?inv-y (make-instance of interval
    (a (/ 1 (send ?y get-b)))
    (b (/ 1 (send ?y get-a)))
    ))
  (* ?x ?inv-y)
)
```

2.5 R

2.5.1 Классы S3

```
# as.interval(c()) — [0, 0]
# as.interval(c(1)) или as.interval(1) — [1, 1]
# as.interval(c(2,3)) — [2, 3]
# as.interval(c(2,3,4,...)) — сообщение об ошибке (слишком большая длина аргумента)
# as.interval(x), где x — интервал, возвращает копию x
as.interval <- function(x)
{
  if (!inherits(x, 'interval'))
  {
    n = length(x)
    if (n == 0) res <- c(0,0)
    if (n == 1) res <- c(x,x)
    if (n == 2) res <- x
    if (n>2) stop("arg's length is greater than 2")
```

```
    class(res) <- c('interval', class(x))
  }
  else res <- x

  res
}
# проверка, является ли x интервалом
is.interval <- function(x) inherits(x, 'interval')

# печать интервала x= [x_1, x_2]
# если x_1 = x_2, то печатается x_1, иначе — [x_1, x_2]
print.interval <- function(x)
{
  if (x[1]==x[2]) cat(x[1],"\n") else cat("[",x[1],x[2],"]\n")
}

# арифметические операции
'+.interval' <- function(x,y)
{
  x1 <- as.interval(x)
  x2 <- as.interval(y)
  as.interval(c(x1[1]+x2[1],x1[2]+x2[2]))
}
'-.interval' <- function(x,y)
{
  x1 <- as.interval(x)
  x2 <- as.interval(y)
  as.interval(c(x1[1]-x2[2],x1[2]-x2[1]))
}
'*.interval' <- function(x,y)
{
  x1 <- as.interval(x)
  x2 <- as.interval(y)
  t <- c(x1[1]*x2[1],x1[1]*x2[2],x1[2]*x2[1],x1[2]*x2[2])
  as.interval(c(min(t),max(t)))
}
'/.interval' <- function(x,y)
{
  x1 <- as.interval(x)
  x2 <- as.interval(y)
  if (x2[1]<=0 && 0<=x2[2]) stop('division␣by␣zero')
  as.interval(x1*c(1/x2[2],1/x2[1]))
}
```

2.5.2 Классы S4

```
Interval <- setClass(
  # Название класса
  "Interval",
  # Поля класса
  slots = c(
    bounds = "numeric"),
  # Значения полей по умолчанию
  prototype = list(
    bounds = c(0.0,0,0)),
```

```
  # Проверка правильности значений границ
  validity = function(object)
  {
    if (object@bounds[1] > object@bounds[2])
    {
      return("Lower bound is greater than upper.")
    }
    return(TRUE)
  }
)

# Обобщённая функция для создания интервалов
setGeneric(name="interval",
  def = function(x) { standardGeneric("interval") }
)
setMethod(f="interval",
  signature="numeric",
  definition = function(x)
  {
    n = length(x)
    if (n == 1) res <- Interval(bounds=c(x,x))
    else
      if (n == 2) res <- Interval(bounds=x)
      else stop("Bad arg's length")
    return(res)
  }
)
setMethod(f="interval",
  signature="Interval",
  definition = function(x) { return(x) }
)

# Вывод интервала на экран
setMethod("show","Interval",
  function(object)
  {
    if (object@bounds[1] == object@bounds[2])
      cat(object@bounds[1],"\n")
    else
      cat("[",object@bounds[1],object@bounds[2],"]\n")
  }
)

# Интервальная арифметика
setMethod("+",c("Interval","Interval"),
  function(e1,e2) {
    bnd <- e1@bounds+e2@bounds
    interval(bnd)
  }
)
setMethod("-",c("Interval","Interval"),
  function(e1,e2)
  {
    lb <- e1@bounds[1]-e2@bounds[2]
    ub <- e1@bounds[2]-e2@bounds[1]
```

```
    interval(c(lb,ub))
  }
)
setMethod("*",c("Interval","Interval"),
  function(e1,e2)
  {
    t <- c()
    for (i in 1:2) for (j in 1:2) t <- c(t,e1@bounds[i]*e2@bounds[j])
    interval(c(min(t),max(t)))
  }
)
setMethod("/",c("Interval","Interval"),
  function(e1,e2)
  {
    if (e2@bounds[1]<=0 && 0<=e2@bounds[2]) stop("Division by zero")
    e1 * interval(c(1/e2@bounds[2],1/e2@bounds[1]))
  }
)
```

3 Примеры использования

3.1 Оценка реальной процентной ставки

Значение реальной процентной ставки (real interest rate, RIR) описывается выражением

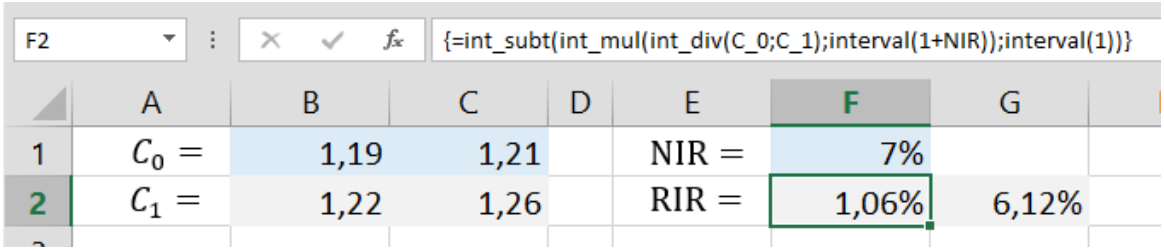
$$\mathrm{RIR} = \frac{C_0}{C_1}(1 + \mathrm{NIR}) - 1,$$

где NIR — номинальная процентная ставка (nominal interest rate) за рассматриваемый период (обычно год), C_0 и C_1 — индексы потребительских цен в начале и в конце периода соответственно.

Требуется оценить значение реальной процентной ставки при номинальной процентной ставке 7% когда индексы потребительских цен в начале C_0 и конце C_0 рассматриваемого периода равны

1. $C_0 = 1.2$ и $C_1 = 1.24$ — числовой результат;
2. $C_0 = [1.19, 1.21]$ и $C_1 = [1.22, 1.26]$ — интервальный результат.

3.1.1 Excel и Calc



3.1.2 Octave

```
C0 = interval(1.19,1.21);
C1 = interval(1.22,1.26);
NIR = 0.07;
RIR = C0/C1*(1+NIR)-1
```

3.1.3 C++

```
cout.precision(3);

MATH::interval_t C0(1.19, 1.21); // C_0 ∈ [1.19;1.21]
MATH::interval_t C1(1.22, 1.26); // C_1 ∈ [1.22;1.26]
double NIR = 0.07;

MATH::interval_t RIR = C0/C1 * interval_t(NIR + 1) - interval_t(1);
cout << "RIR is in [" << RIR.lo() * 100 << "% ; "
                      << RIR.hi() * 100 << "%]" << endl;
```

или

```
cout.precision(3);

MATH::interval_t<> C0(119, 100, 121, 100); // C_0 ∈ [1.19;1.21]
MATH::interval_t<> C1(122, 100, 126, 100); // C_1 ∈ [1.22;1.26]
MATH::interval_t<> NIR(7, 100);            // NIR = 7%

MATH::interval_t<> RIR = C0/C1 * (NIR + 1) - 1;
cout << "RIR is in [" << RIR.lo() * 100 << "% ; "
                      << RIR.hi() * 100 << "%]" << endl;
```

3.1.4 CLIPS

```
(bind ?C0 (create-interval  1.19 1.21))
(bind ?C1 (create-interval  1.22 1.26))
(bind ?NIR (create-interval 0.07))
(bind ?one (create-interval 1))
; оценка RIR
(bind ?RIR (- (* (/ ?C0 ?C1) (+ ?one ?NIR)) ?one))
; удаление временных объектов
(send ?RIR put-temporary F)
(clear-temporary-garbage)
; вывод результата на экран
(printout t "RIR = " (send ?RIR show) crlf)
```

3.1.5 R

Классы S3:

```
NIR <- 0.07
C0 <- as.interval(c(1.19,1.21))
C1 <- as.interval(c(1.22,1.26))

# RIR = C0/C1*(1+NIR)-1
C0/C1*(1+NIR)-1

# ...или RIR ≈ NIR - (C1-C0)/C0
NIR-(C1-C0)/C0
```

Классы S4:

```
NIR <- 0.07
C0 <- Interval(bounds=c(1.19,1.21))
```

```
C1 <- interval(c(1.22,1.26))

# RIR = C0/C1(1 + NIR) - 1
C0/C1*interval(1+NIR)-interval(1)

# ...или RIR ≈ NIR - (C1-C0)/C0
interval(NIR)-(C1-C0)/C0
```

Список литературы

[1] *Алефельд, Г.* Введение в интервальные вычисления: Пер. с англ. / Г. Алефельд, Ю. Херцбергер. — М.: Мир, 1987. — 360 с.