Python

unknown

сент. 07, 2023

Оглавление

1 Contents **3**
1.1 Installation . . . 3
1.2 Using intervals . . . 3
1.3 General purpose functions . . . 7
1.4 Recognizing functionals . . . 16
1.5 Interval linear systems . . . 21

Welcome to the IntvalPy documentation! This Python module implements an algebraically closed system for working with intervals, and provides the ability to work with both classical interval arithmetic and full Kaucher interval arithmetic. The top-level functionality of the IntvalPy library implements the latest methods for recognizing and evaluating sets of solutions to interval linear systems of equations, calculating their formal solutions, and visualizing sets of solutions to interval equations and systems of equations.

Глава 1

Contents

1.1 Installation

The most convenient way to install this library is to use the **pip_** command:

```
pip install --upgrade pip
pip install intvalpy
```

1.1.1 Installation from the source codes

You can either download the sources from PyPI or clone the repository hosted on GitHub:

```
git clone https://github.com/AndrosovAS/intvalpy.git
cd intvalpy
python setup.py install
```

1.2 Using intervals

This section gives an overview of the use of interval classes and examples of working with interval data.

Run the following commands to connect the necessary modules

```
>>> import numpy as np
>>> from intvalpy import Interval, precision
```

It connects the interval function *Interval* from the implemented *IntvalPy* library, which was previously installed with the command *pip install intvalpy*.

Содержание

- *Using intervals*
  - *Basic characteristics*
  - *Interval vectors and matrices*
  - *Create intervals*

The class *Interval* was created in accordance with IEEE Standard 754-2008, where rounding occurs to the nearest even number. This allows significantly to accelerate the computational upper-level functions and reduce the computation time. However, in tasks where you need more precision, you can switch from the standard representation of the number as *double float* type to *mpf* type. To do this, run the following command:

```
>>> precision.extendedPrecisionQ = True
```

You can also set the working precision (after which decimal place rounding will take place):

```
>>> precision.dps(50)
```

The default setting is increased accuracy to 36th decimal place.

The IntvalPy library supports classical arithmetic and full Kaucher interval arithmetic. Each arithmetic has different types of intervals (for example, classical arithmetic considers only «correct» intervals), which means that the arithmetic operations can differ from each other. Therefore, it was decided to develop two different classes for each of the arithmetics. However, there are not few common characteristics that these classes could inherit from some third class. These could be the operations of taking the ends of an interval, the width or radius, and many other things. This is why the parent class ``BaseTools““ was made separately.

1.2.1 Basic characteristics

class BaseTools(left, right)

A parent class that contains methods that can be used to calculate the basic interval characteristics of any interval arithmetic. Used in the *ClassicalArithmetic* and *KaucherArithmetic* classes.

Parameters:

- **left**
  [int, float] The lower (left) limit of the interval.
- **right**
  [int, float] The upper (right) limit of the interval.

Methods:

1. a, inf: The operation of taking the lower (left) bound of the interval.
2. b, sup: The operation of taking the upper (right) bound of the interval.
3. copy: Creates a deep copy of the interval.
4. to_float: If increased accuracy is enabled, it is sometimes necessary to convert to standard accuracy (float64)
5. wid: Width of the non-empty interval.
6. rad: Radius of the non-empty interval.

7. mid: Midpoint of the non-empty interval.
8. mig: The smallest absolute value in the non-empty interval.
9. mag: The greatest absolute value in the non-empty interval.
10. dual: Dual interval.
11. pro: Correct projection of an interval.
12. opp: Algebraically opposite interval.
13. inv: Algebraically opposite interval.
14. khi: Ratschek's functional of an interval.

1.2.2 Interval vectors and matrices

class ArrayInterval(intervals)

It is often necessary to consider intervals not as separate objects, but as interval vectors or matrices. It is important to note that different intervals can be from different interval arithmetics, which leads to additional checks when performing arithmetic operations. The IntvalPy library, using the ArrayInterval class, allows you to create such arrays of any nesting. In fact, we use arrays created using the *numpy* library.

This class uses all the methods of the BaseTools class, but there are additional features that are common to interval vectors and matrices.

Parameters:

- **intervals**
  [ndarray] The numpy array with objects of type ClassicalArithmetic and KaucherArithmetic.

Methods:

1. data: An array of interval data of type *ndarray*.
2. shape: The elements of the shape tuple give the lengths of the corresponding interval array dimensions.
3. ndim: Number of interval array dimensions.
4. ranges: A list of indexes for each dimension.
5. vertex: The set of extreme points of an interval vector.
6. T: View of the transposed interval array.
7. reshape(new_shape): Gives a new shape to an interval array without changing its data.

Examples:

Matrix product

```
>>> f = Interval([
      [[-1, 3], [-2, 5]],
      [[-7, -4], [-5, 7]]
    ])
>>> s = Interval([
      [[-3, -2], [4, 4]],
      [[-7, 3], [-8, 0]]
    ])
>>> f @ s
```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Interval([['[-44.0, 18.0]', '[-44.0, 28.0]']
            ['[-41.0, 56.0]', '[-84.0, 24.0]']])
```

Transpose

```
>>> f.T
# Interval([['[-1, 3]', '[-7, -4]'],
            ['[-2, 5]', '[-5, 7]']])
```

1.2.3 Create intervals

def Interval(*args, sortQ=True, midRadQ=False)

When creating an interval, you must consider which interval arithmetic it belongs to, and how it is defined: by means of the left and right values, through the middle and radius, or as a single object. For this purpose, a universal function *Interval* has been implemented, which can take into account all the aspects described above. In addition, it has a parameter for automatic conversion of the ends of an interval, so that when the user creates it, he can be sure, that he works with the classical type of intervals.

Parameters:

- **args**
  [int, float, list, ndarray] If the argument is a single one, then the intervals are set as single objects. To do this you must create array, each element of which is an ordered pair of the lower and upper bound of the interval.

  If the arguments are two, then the flag of the *midRadQ* parameter is taken into account. If the value is *True*, then the interval is set through the middle of the interval and its radius. Otherwise, the first argument will stand for the lower ends, and the second argument the upper ends.
- **sortQ**
  [bool, optional] Parameter determines whether the automatic conversion of the interval ends should be performed. The default is *True*.
- **midRadQ**
  [bool, optional] The parameter defines whether the interval is set through its middle and radius. The default is *False*.

Examples:

Creating intervals by specifying arrays of left and right ends of intervals

```
>>> a = [2, 5, -3]
>>> b = [4, 7, 1]
>>> Interval(a, b)
# Interval(['[2, 4]', '[5, 7]', '[-3, 1]'])
```

Now let's create the same interval vector, but in a different way

```
>>> Interval([ [2, 4], [5, 7], [-3, 1] ])
# Interval(['[2, 4]', '[5, 7]', '[-3, 1]'])
```

In case it is necessary to work with an interval object from Kaucher arithmetic, it is necessary to disable automatic converting ends

```
>>> Interval(5, -2, sortQ=False)
# '[5, -2]'
```

As mentioned earlier, the IntvalPy library allows you to work with vectors and matrices. This automatically generates the need to calculate the length of the array, as well as the possibility of working with collections.

```
>>> f = Interval([ [2, 4], [5, 7], [-3, 1] ])
>>> len(f)
# 3
```

To get the N-th value or several values (in the future we will call it a slice of the array) you can use quite usual tools. Moreover, since the class *ArrayInterval* is changeable, it is also possible to change or delete elements:

```
>>> f[1]
# [5, 7]
>>> f[1:]
# Interval(['[5, 7]', '[-3, 1]'])
>>> f[1:] = Interval([ [-5, 5], [-10, 10] ])
>>> f
# Interval(['[2, 4]', '[-5, 5]', '[-10, 10]'])
>>> del f[1]
>>> f
# Interval(['[2, 4]', '[-10, 10]'])
```

1.3 General purpose functions

In this section, we present an overview of functions for working with interval quantities as well as some functions for creating interval objects.

Run the following commands to connect the necessary modules

```
>>> import intvalpy as ip
>>> import numpy as np
```

Содержание

- *General purpose functions*
  - *Converting data to interval type*
  - *Interval scatterplot*
  - *Intersection of intervals*
  - *Distance*
  - *Zero intervals*
  - *Identity interval matrix*
  - *Diagonal of the interval matrix*
  - *Elementary mathematical functions*

* *The square root*
* *The exponent*
* *The natural logarithm*
* *The sine function*
* *The cosine function*

– *Test interval systems*

* *The Shary system*
* *The Neumaier-Reichmann system*
* *References*

1.3.1 Converting data to interval type

def asinterval(a)

To convert the input data to the interval type, use the *asinterval* function:

Parameters:

- **a**
  [int, float, array_like] Input data in any form that can be converted to an interval data type. These include int, float, list and ndarrays.

Returns:

- **out**
  [Interval] The conversion is not performed if the input is already of type Interval. Otherwise an object of interval type is returned.

Examples:

```
>>> ip.asinterval(3)
'[3, 3]'
>>> ip.asinterval([1/2, ip.Interval(-2, 5), 2])
Interval(['[0.5, 0.5]', '[-2, 5]', '[2, 2]'])
```

1.3.2 Interval scatterplot

Математическая диаграмма, изображающая значения двух переменных в виде брусов на декартовой плоскости.

Parameters:

- **x**
  [Interval] Интервальный вектор положения данных на оси OX.
- **y**
  [Interval] Интервальный вектор положения данных на оси OY.
- **title: str, optional**
  Верхняя легенда графика.

- **color: str, optional**
  Цвет отображения брусов.
- **alpha: float, optional**
  Прозрачность брусов.
- **s: float, optional**
  Насколько велики точки вершин.
- **size: tuple, optional**
  Размер отрисовочного окна.
- **save: bool, optional**
  Если значение True, то график сохраняется.

Returns:

- **out: None**
  A scatterplot is displayed.

Examples:

```
>>> x = ip.Interval(np.array([1.06978355, 1.94152571, 1.70930717, 2.94775725, 4.55556349,
↪ 6, 6.34679035, 6.62305275]), \
>>                 np.array([1.1746937 , 2.73256075, 1.95913956, 3.61482169, 5.40818299,
↪ 6, 7.06625362, 7.54738552]))
>>> y = ip.Interval(np.array([0.3715678 , 0.37954135, 0.38124681, 0.39739009, 0.42010472,
↪ 0.45, 0.44676075, 0.44823645]), \
>>                 np.array([0.3756708 , 0.4099036 , 0.3909104 , 0.42261893, 0.45150898,
↪ 0.45, 0.47255936, 0.48118948]))
>>> ip.scatter_plot(x, y)
```

1.3.3 Intersection of intervals

Функция `intersection` осуществляет пересечение интервальных данных. В случае, если на вход поданы массивы, то осуществляется покомпонентное пересечение.

Parameters:

A, B: `Interval`
В случае, если операнды не являются интервальным типом, то они преобразуются функцией `asinterval`.

Returns:

out: `Interval`
Возвращается массив пересечённых интервалов. Если некоторые интервалы не пересекаются, то на их месте выводится интервал `Interval(float('-inf'), float('-inf'))`.

Примеры:

```
>>> import intvalpy as ip
>>> f = ip.Interval([-3., -6., -2.], [0., 5., 6.])
>>> s = ip.Interval(-1, 10)
>>> ip.intersection(f, s)
interval(['[-1.0, 0.0]', '[-1.0, 5.0]', '[-1.0, 6.0]'])
```

```
>>> f = ip.Interval([-3., -6., -2.], [0., 5., 6.])
>>> s = -2
>>> ip.intersection(f, s)
interval(['[-2.0, -2.0]', '[-2.0, -2.0]', '[-2.0, -2.0]'])
```

```
>>> f = ip.Interval([-3., -6., -2.], [0., 5., 6.])
>>> s = ip.Interval([ 2., -8., -6.], [6., 7., 0.])
>>> ip.intersection(f, s)
interval(['[-inf, -inf]', '[-6.0, 5.0]', '[-2.0, 0.0]'])
```

1.3.4 Distance

def dist(x, y, order=float(„inf“))

To calculate metrics or multimetrics in interval spaces, the *dist* function is provided. The mathematical formula for distance is given as follows: $\mathrm{dist}_{\mathrm{order}} = (\mathrm{sum}_{\mathrm{ij}} \, ||\mathrm{x}_{\mathrm{ij}} - \mathrm{y}_{\mathrm{ij}}||^{\mathrm{order}})^{1/\mathrm{order}}$.

It is important to note that this formula involves an algebraic difference, not the usual interval difference.

Parameters:

- **a, b**
  [Interval] The intervals between which you need to calculate the distance. In the case of multidimensional operands a multimetric is calculated.
- **order**
  [int, optional] The order of the metric is set. By default, setting is Chebyshev distance.

Returns:

- **out: float**
  The distance between the input operands is returned.

Examples:

```
>>> f = ip.Interval([
        [[0, 1], [2, 3]],
        [[4, 5], [6, 7]],
    ])
>>> s = ip.Interval([
        [[1, 2], [3, 4]],
        [[5, 6], [7, 8]],
    ])
>>> ip.dist(f, s)
1.0
```

The detailed information about various metrics can be found in the referenced monograph.

1.3.5 Zero intervals

def zeros(shape)

To create an interval array where each element is point and equal to zero, the function *zeros* is provided:

Parameters:

- **shape**
  [int, tuple] Shape of the new interval array, e.g., (2, 3) or 4.

Returns:

- **out**
  [Interval] An interval array of zeros with a given shape

Examples:

```
>>> ip.zeros((2, 3))
Interval([['[0, 0]', '[0, 0]', '[0, 0]'],
          ['[0, 0]', '[0, 0]', '[0, 0]']])
>>> ip.zeros(4)
Interval(['[0, 0]', '[0, 0]', '[0, 0]', '[0, 0]'])
```

1.3.6 Identity interval matrix

def eye(N, M=None, k=0)

Return a 2-D interval array with ones on the diagonal and zeros elsewhere.

Parameters:

- **N**
  [int] Shape of the new interval array, e.g., (2, 3) or 4.
- **M**
  [int, optional] Number of columns in the output. By default, M = N.
- **k**
  [int, optional] Index of the diagonal: 0 refers to the main diagonal, a positive value refers to an upper diagonal, and a negative value to a lower diagonal. By default, k = 0.

Returns:

- **out**
  [Interval of shape (N, M)] An interval array where all elements are equal to zero, except for the k-th diagonal, whose values are equal to one.

Examples:

```
>>> ip.eye(3, M=2, k=-1)
Interval([['[0, 0]', '[0, 0]'],
          ['[1, 1]', '[0, 0]'],
          ['[0, 0]', '[1, 1]']])
```

1.3.7 Diagonal of the interval matrix

def diag(v, k=0)

Extract a diagonal or construct a diagonal interval array.

Parameters:

- **v**
  [Interval] If v is a 2-D interval array, return a copy of its k-th diagonal. If v is a 1-D interval array, return a 2-D interval array with v on the k-th diagonal.
- **k**
  [int, optional] Diagonal in question. Use k>0 for diagonals above the main diagonal, and k<0 for diagonals below the main diagonal. By default, k=0.

Returns:

- **out**
  [Interval] The extracted diagonal or constructed diagonal interval array.

Examples:

```
>>> A, b = ip.Shary(3)
>>> ip.diag(A)
Interval(['[2, 3]', '[2, 3]', '[2, 3]'])
>>> ip.diag(b)
Interval([['[-2, 2]', '[0, 0]', '[0, 0]'],
          ['[0, 0]', '[-2, 2]', '[0, 0]'],
          ['[0, 0]', '[0, 0]', '[-2, 2]']])
```

1.3.8 Elementary mathematical functions

This section presents the basic elementary mathematical functions that are most commonly encountered in various kinds of applied problems.

The square root

def sqrt(x)

Interval enclosure of the square root intrinsic over an interval.

Parameters:

- **x**
  [Interval] The values whose square-roots are required.

Returns:

- **out**
  [Interval] An array of the same shape as x, containing the interval enclosure of the square root of each element in x.

Examples:

```
>>> f = ip.Interval([[-3, -1], [-3, 2], [0, 4]])
>>> ip.sqrt(f)
Interval(['[nan, nan]', '[0, 1.41421]', '[0, 2]'])
```

The exponent

def exp(x)

Interval enclosure of the exponential intrinsic over an interval.

Parameters:

- **x**
  [Interval] The values to take the exponent from.

Returns:

- **out**
  [Interval] An array of the same shape as x, containing the interval enclosure of the exponential of each element in x.

Examples:

```
>>> f = ip.Interval([[-3, -1], [-3, 2], [0, 4]])
>>> ip.exp(f)
Interval(['[0.0497871, 0.367879]', '[0.0497871, 7.38906]', '[1, 54.5982]'])
```

The natural logarithm

def log(x)

Interval enclosure of the natural logarithm intrinsic over an interval.

Parameters:

- **x**
  [Interval] The values to take the natural logarithm from.

Returns:

- **out**
  [Interval] An array of the same shape as x, containing the interval enclosure of the natural logarithm of each element in x.

Examples:

```
>>> f = ip.Interval([[-3, -1], [-3, 2], [1, 4]])
>>> ip.log(f)
Interval(['[nan, nan]', '[-inf, 0.693147]', '[0, 1.38629]'])
```

The sine function

def sin(x)

Interval enclosure of the sin intrinsic over an interval.

Parameters:

- **x**
  [Interval] The values to take the sin from.

Returns:

- **out**
  [Interval] An array of the same shape as x, containing the interval enclosure of the sin of each element in x.

Examples:

```
>>> f = ip.Interval([[-3, -1], [-3, 2], [0, 4]])
>>> ip.sin(f)
Interval(['[-1, -0.14112]', '[-1, 1]', '[-0.756802, 1]'])
```

The cosine function

def cos(x)

Interval enclosure of the cos intrinsic over an interval.

Parameters:

- **x**
  [Interval] The values to take the cos from.

Returns:

- **out**
  [Interval] An array of the same shape as x, containing the interval enclosure of the cos of each element in x.

Examples:

```
>>> f = ip.Interval([[-3, -1], [-3, 2], [0, 4]])
>>> ip.cos(f)
Interval(['[-0.989992, 0.540302]', '[-0.989992, 1]', '[-1, 1]'])
```

1.3.9 Test interval systems

To check the performance of each implemented algorithm, it is tested on well-studied test systems. This subsection describes some of these systems, for which the properties of the solution sets are known, and their analytical characteristics and the complexity of numerical procedures have been previously studied.

The Shary system

def Shary(n, N=None, alpha=0.23, beta=0.35)

One of the popular test systems is the Shary system. Due to its symmetry, it is quite simple to determine the structure of its united solution set as well as other solution sets. Changing the values of the system parameters, you can get an extensive family of interval linear systems for testing the numerical algorithms. As the parameter beta decreases, the matrix of the system becomes more and more singular, and the united solution set enlarges indefinitely.

Parameters:

- **n**
  [int] Dimension of the interval system. It may be greater than or equal to two.
- **N**
  [float, optional] A real number not less than (n 1). By default, N = n.

- **alpha**
  [float, optional] A parameter used for specifying the lower endpoints of the elements in the interval matrix. The parameter is limited to 0 < alpha <= beta <= 1. By default, alpha = 0.23.
- **beta**
  [float, optional] A parameter used for specifying the upper endpoints of the elements in the interval matrix. The parameter is limited to 0 < alpha <= beta <= 1. By default, beta = 0.35.

Returns:

- **out: Interval, tuple**
  The interval matrix and interval vector of the right side are returned, respectively.

Examples:

```
>>> A, b = ip.Shary(3)
>>> print('A: ', A)
>>> print('b: ', b)
A:  Interval([['[2, 3]', '[-0.77, 0.65]', '[-0.77, 0.65]'],
          ['[-0.77, 0.65]', '[2, 3]', '[-0.77, 0.65]'],
          ['[-0.77, 0.65]', '[-0.77, 0.65]', '[2, 3]']])
b:  Interval(['[-2, 2]', '[-2, 2]', '[-2, 2]'])
```

The Neumaier-Reichmann system

def Neumaier(n, theta, infb=None, supb=None)

This system is a parametric interval linear system, first proposed by K. Reichmann [2], and then slightly modified by A. Neumaier. The matrix of the system can be regular, but not strongly regular for some values of the diagonal parameter. It is shown that n × n matrices are non-singular for theta > n provided that n is even, and, for odd order n, the matrices are non-singular for theta > sqrt(n^2 - 1).

Parameters:

- **n**
  [int] Dimension of the interval system. It may be greater than or equal to two.
- **theta**
  [float, optional] Nonnegative real parameter, which is the number that stands on the main diagonal of the matrix A.
- **infb**
  [float, optional] A real parameter that specifies the lower endpoints of the components of the right-hand side vector. By default, infb = -1.
- **supb**
  [float, optional] A real parameter that specifies the upper endpoints of the components of the right-hand side vector. By default, supb = 1.

Returns:

- **out: Interval, tuple**
  The interval matrix and interval vector of the right side are returned, respectively.

Examples:

```
>>> A, b = ip.Neumaier(2, 3.5)
>>> print('A: ', A)
>>> print('b: ', b)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
A:  Interval([['[3.5, 3.5]', '[0, 2]'],
          ['[0, 2]', '[3.5, 3.5]']])
b:  Interval(['[-1, 1]', '[-1, 1]'])
```

References

[1] S.P. Shary - On optimal solution of interval linear equations // SIAM Journal on Numerical Analysis. – 1995. – Vol. 32, No. 2. – P. 68–630.

[2] Reichmann K. Abbruch beim Intervall-Gauß-Algorithmus // Computing. – 1979. – Vol. 22, Issue 4. – P. 355–361.

[3] С.П. Шарый - Конечномерный интервальный анализ.
Sergey P. Shary, **`Finite-Dimensional Interval Analysis` _**.

1.4 Recognizing functionals

This paragraph presents an overview of functions for investigating solvability of the set of solutions of interval linear systems.

Run the following commands to connect the necessary modules

```
>>> import intvalpy as ip
>>> ip.precision.extendedPrecisionQ = False
>>> import numpy as np
```

Before we start solving a system of equations with interval data it is necessary to understand whether it is solvable or not. To do this we consider the problem of decidability recognition, i.e. non-emptiness of the set of solutions. In the case of an interval linear (m x n)-system of equations, we will need to solve no more than 2^n linear inequalities of size 2m+n. This follows from the fact of convexity and polyhedra of the intersection of the sets of solutions interval system of linear algebraic equations (ISLAU) with each of the orthants of $\mathbf{R}^n$ space. Reducing the number of inequalities is fundamentally impossible, which follows from the fact that the problem is intractable, i.e. its NP-hardness. It is clear that the above described method is applicable only for small dimensionality of the problem, that is why the *recognizing functional method* was proposed.

1.4.1 Tol for linear systems

class Tol

To check the interval system of linear equations for its strong compatibility, the recognizing functional Tol should be used.

Value at the point

** class Tol.value(A, b, x, weight=None)**

The function computes the value of the recognizing functional at the point x.

Parameters:

- **A: Interval**
  The input interval matrix of ISLAE, which can be either square or rectangular.
- **b: Interval**
  The interval vector of the right part of the ISLAE.
- **x: np.array, optional**
  The point at which the recognizing functional is calculated.
- **weight: float, np.array, optional**
  The vector of weight coefficients for each forming of the recognizing functional. By default, it is a vector consisting of ones.

Returns:

- **out: float**
  The value of the recognizing functional at the point x.

Examples:

As an example, consider the well-known interval system proposed by Barth-Nuding:

```
>>> A = ip.Interval([
        [[2, 4], [-1, 2]],
        [[-2, 1], [2, 4]]
    ])
>>> b = ip.Interval([[-2, 2], [-2, 2]])
```

To get the value of a function at a specific point, perform the following instruction

```
>>> x = np.array([1, 2])
>>> ip.linear.Tol.value(A, b, x)
-7.0
```

The point x does not lie in the tolerable solution set for this system, because the value of the recognizing functional is negative.

Finding a global maximum

** class Tol.maximize(A, b, x0=None, weight=None, linear_constraint=None, kwargs)**

The function is intended for finding the global maximum of the recognizing functional. The ralgb5 subgradient method is used for optimization.

Parameters:

- **A: Interval**
  The input interval matrix of ISLAE, which can be either square or rectangular.
- **b: Interval**
  The interval vector of the right part of the ISLAE.

- **x0: np.array, optional**
  The initial assumption is at what point the maximum is reached. By default, x0 is equal to the vector which is the solution (pseudo-solution) of the system mid(A) x = mid(b).
- **weight: float, np.array, optional**
  The vector of weight coefficients for each forming of the recognizing functional. By default, it is a vector consisting of ones.
- **linear_constraint: LinearConstraint, optional**
  System (lb <= C <= ub) describing linear dependence between parameters. By default, the problem of unconditional maximization is being solved.
- **kwargs: optional params**
  The ralgb5 function uses additional parameters to adjust its performance. These parameters include the step size, the stopping criteria, the maximum number of iterations and others. Specified in the function description ralgb5.

Returns:

- **out: tuple**
  The function returns the following values in the specified order: 1. the vector solution at which the recognition functional reaches its maximum, 2. the value of the recognition functional, 3. the number of iterations taken by the algorithm, 4. the number of calls to the calcfg function, 5. the exit code of the algorithm (1 = tolf, 2 = tolg, 3 = tolx, 4 = maxiter, 5 = error).

Examples:

To identify whether the data is strong compatibility, optimization must be performed:

```
>>> A = ip.Interval([
        [[2, 4], [-1, 2]],
        [[-2, 1], [2, 4]]
    ])
>>> b = ip.Interval([[-2, 2], [-2, 2]])
>>> ip.linear.Tol.maximize(A, b)
(array([0., 0.]), 2.0, 29, 46, 1)
```

The distinguishing feature of the *Tol* functional from the *Uni* and *Uss* functional is that regardless of whether the matrix **A** interval or point matrix, the functional always has only one extremum. Thus it does not matter which initial guess to start the search with. However, if one specifies an initial point, the search for a global maximum can be accelerated.

In addition, conditional optimization with linear constraints has been implemented using the penalty function method.

```
>>> A = ip.Interval([
        [[2, 4], [10, 11.99999]],
        [[-2, 1], [2, 4]]
    ])
>>> b = ip.Interval([[-2, 2], [-2, 2]]) + 0.15
```

```
>>> C = np.array([
        [1, 0],
        [0, 1]
    ])
>>> ub = np.array([5, 5])
>>> lb = np.array([0, 0.1])
```

```
>>> linear_constraint = ip.LinearConstraint(C, ub=ub, lb=lb)
>>> ip.linear.Tol.maximize(A, b, linear_constraint=linear_constraint, tolx=1e-20)
(array([3.48316025e-17, 1.00000000e-01]), 0.9500009999999999, 114, 288, 1)
```

1.4.2 Uni for linear systems

class Uni

To check the interval system of linear equations for its weak compatibility, the recognizing functional Uni should be used.

Value at the point

** class Uni.value(A, b, x, weight=None)**

The function computes the value of the recognizing functional at the point x.

Parameters:

- **A: Interval**
  The input interval matrix of ISLAE, which can be either square or rectangular.
- **b: Interval**
  The interval vector of the right part of the ISLAE.
- **x: np.array, optional**
  The point at which the recognizing functional is calculated.
- **weight: float, np.array, optional**
  The vector of weight coefficients for each forming of the recognizing functional. By default, it is a vector consisting of ones.

Returns:

- **out: float**
  The value of the recognizing functional at the point x.

Examples:

As an example, consider the well-known interval system proposed by Barth-Nuding:

```
>>> A = ip.Interval([
        [[2, 4], [-1, 2]],
        [[-2, 1], [2, 4]]
    ])
>>> b = ip.Interval([[-2, 2], [-2, 2]])
```

To get the value of a function at a specific point, perform the following instruction

```
>>> x = np.array([1, 2])
>>> ip.linear.Uni.value(A, b, x)
0.0
```

The point x does lie in the united solution set for this system, because the value of the recognizing functional is not negative.

Finding a global maximum

** class Uni.maximize(A, b, x0=None, weight=None, linear_constraint=None, kwargs)**

The function is intended for finding the global maximum of the recognizing functional. The ralgb5 subgradient method is used for optimization.

Parameters:

- **A: Interval**
  The input interval matrix of ISLAE, which can be either square or rectangular.
- **b: Interval**
  The interval vector of the right part of the ISLAE.
- **x0: np.array, optional**
  The initial assumption is at what point the maximum is reached. By default, x0 is equal to the vector which is the solution (pseudo-solution) of the system mid(A) x = mid(b).
- **weight: float, np.array, optional**
  The vector of weight coefficients for each forming of the recognizing functional. By default, it is a vector consisting of ones.
- **linear_constraint: LinearConstraint, optional**
  System (lb <= C <= ub) describing linear dependence between parameters. By default, the problem of unconditional maximization is being solved.
- **kwargs: optional params**
  The ralgb5 function uses additional parameters to adjust its performance. These parameters include the step size, the stopping criteria, the maximum number of iterations and others. Specified in the function description ralgb5.

Returns:

- **out: tuple**
  The function returns the following values in the specified order: 1. the vector solution at which the recognition functional reaches its maximum, 2. the value of the recognition functional, 3. the number of iterations taken by the algorithm, 4. the number of calls to the calcfg function, 5. the exit code of the algorithm (1 = tolf, 2 = tolg, 3 = tolx, 4 = maxiter, 5 = error).

Examples:

To identify whether the data is weak compatibility, optimization must be performed:

```
>>> A = ip.Interval([
        [[2, 4], [-1, 2]],
        [[-2, 1], [2, 4]]
    ])
>>> b = ip.Interval([[-2, 2], [-2, 2]])
>>> ip.linear.Uni.maximize(A, b)
(array([0., 0.]), 2.0, 29, 45, 1)
```

However, we know from theory that even in the linear case the recognizing function Uni is not a concave function on the whole investigated space. Thus, there is no guarantee that the global maximum of the function, and not the local extremum, was found using the optimization algorithm.

As some solution, the user can specify an initial guess, based, for example, on the features of the matrix. This can also speed up the process of finding the global maximum.

In addition, conditional optimization with linear constraints has been implemented using the penalty function method.

```
>>> A = ip.Interval([
        [[2, 4], [10, 11.99999]],
        [[-2, 1], [2, 4]]
    ])
>>> b = ip.Interval([[-2, 2], [-2, 2]]) + 0.15
```

```
>>> C = np.array([
        [1, 0],
        [0, 1]
    ])
>>> ub = np.array([5, 5])
>>> lb = np.array([0, 0.1])
```

```
>>> linear_constraint = ip.LinearConstraint(C, ub=ub, lb=lb)
>>> ip.linear.Uni.maximize(A, b, linear_constraint=linear_constraint, tolx=1e-20)
(array([1.47928518e-17, 1.00000000e-01]), 1.15, 110, 259, 1)
```

References

[1] С.П. Шарый - Разрешимость интервальных линейных уравнений и анализ данных с неопределённостями // Автоматика и Телемеханика, No 2, 2012

[2] С.П. Шарый, И.А. Шарая - Распознавание разрешимости интервальных уравнений и его приложения к анализу данных // Вычислительные технологии, Том 18, No 3, 2013, стр. 80-109.

[3] С.П. Шарый - Сильная согласованность в задаче восстановления зависимостей при интервальной неопределённости данных // Вычислительные технологии, Том 22, No 2, 2017, стр. 150-172.

1.5 Interval linear systems

This paragraph presents an overview of functions for obtaining estimates of the set of solutions of interval linear systems.

Run the following commands to connect the necessary modules

```
>>> import intvalpy as ip
>>> import numpy as np
```

1.5.1 Variability of the solution

def ive(A, b, N=40)

When solving the system, we usually get many different estimates, equally suitable as answers to the problem and consistent with its data. It is the variability that characterizes how small or large this set is.

To get a quantitative measure, use the *ive* function:

Parameters:

- **A**
  [Interval] The input interval matrix of ISLAE, which can be either square or rectangular.

- **b**
  [Interval] The interval vector of the right part of the ISLAE.
- **N**
  [int, optional] The number of corner matrices for which the conditionality is calculated. By default, N = 40.

Returns:

- **out**
  [float] A measure of the variability of an interval system of linear equations IVE.

Examples:

A randomized algorithm was used to speed up the calculations, so there is a chance that a non-optimal value will be found. To overcome this problem we can increase the value of the parameter N.

```
>>> A = ip.Interval([
        [[98, 100], [99, 101]],
        [[97, 99], [98, 100]],
        [[96, 98], [97, 99]]
    ])
>>> b = ip.Interval([[190, 210], [200, 220], [190, 210]])
>>> ip.linear.ive(A, b, N=60)
1.56304
```

For more information, see the article Shary S.P.

1.5.2 Метод граничных интервалов

В случае, когда появляется необходимость визуализировать множество решений системы линейных неравенств (или интервальную систему уравнений), а также получить все вершины множество, можно прибегнуть к методам решения проблемы перечисления вершин. Однако существующие реализации имеют ряд недостатков: работа только с квадратными системами, плохая обработка неограниченных множеств.

Основываясь на применении *матрицы граничных интервалов* был предложен *метод граничных интервалов* для исследования и визуализации полиэдральных множеств. Главными преимуществами данного подхода является возможность работать с неограниченными и тощими множествами решений, а также с линейными системами, когда количество уравнений отлично от количества неизвестных.

Для общего понимания работы алгоритма укажем его основные шаги:

```
1. Формирование матрицы граничных интервалов;
2. Изменение матрицы граничных интервалов с учётом окна отрисовки;
3. Построение упорядоченных вершин полиэдрального множества решений;
4. Вывод построенных вершин и (если надо) отрисовка полиэдра.
```

Двумерная визуализация линейной системы неравенств

Для работы с линейной системой алгебраических неравенств A x >= b, когда количество неизвестных равно двум, необходимо воспользоваться функций `lineqs`. В случае, если множество решений неограниченно, то алгоритм самостоятельно выберет границы отрисовки. Однако пользователь сам может указать их явным образом.

Parameters:

- **A: float**
  Матрица системы линейных алгебраических неравенств.
- **b: float**
  Вектор правой части системы линейных алгебраических неравенств.
- **show: bool, optional**
  Данный параметр отвечает за то будет ли показано множество решений. По умолчанию указано значение True, т.е. происходит отрисовка графика.
- **title: str, optional**
  Верхняя легенда графика.
- **color: str, optional**
  Цвет внутренней области множества решений.
- **bounds: array_like, optional**
  Границы отрисовочного окна. Первый элемент массива отвечает за нижние грани по осям OX и OY, а второй за верхние. Таким образом, для того, чтобы OX лежало в пределах [-2, 2], а OY в пределах [-3, 4], необходимо задать `bounds` как [[-2, -3], [2, 4]].
- **alpha: float, optional**
  Прозрачность графика.
- **s: float, optional**
  Насколько велики точки вершин.
- **size: tuple, optional**
  Размер отрисовочного окна.
- **save: bool, optional**
  Если значение True, то график сохраняется.

Returns:

- **out: list**
  Возвращается список упорядоченных вершин. В случае, если show = True, то график отрисовывается.

Examples:

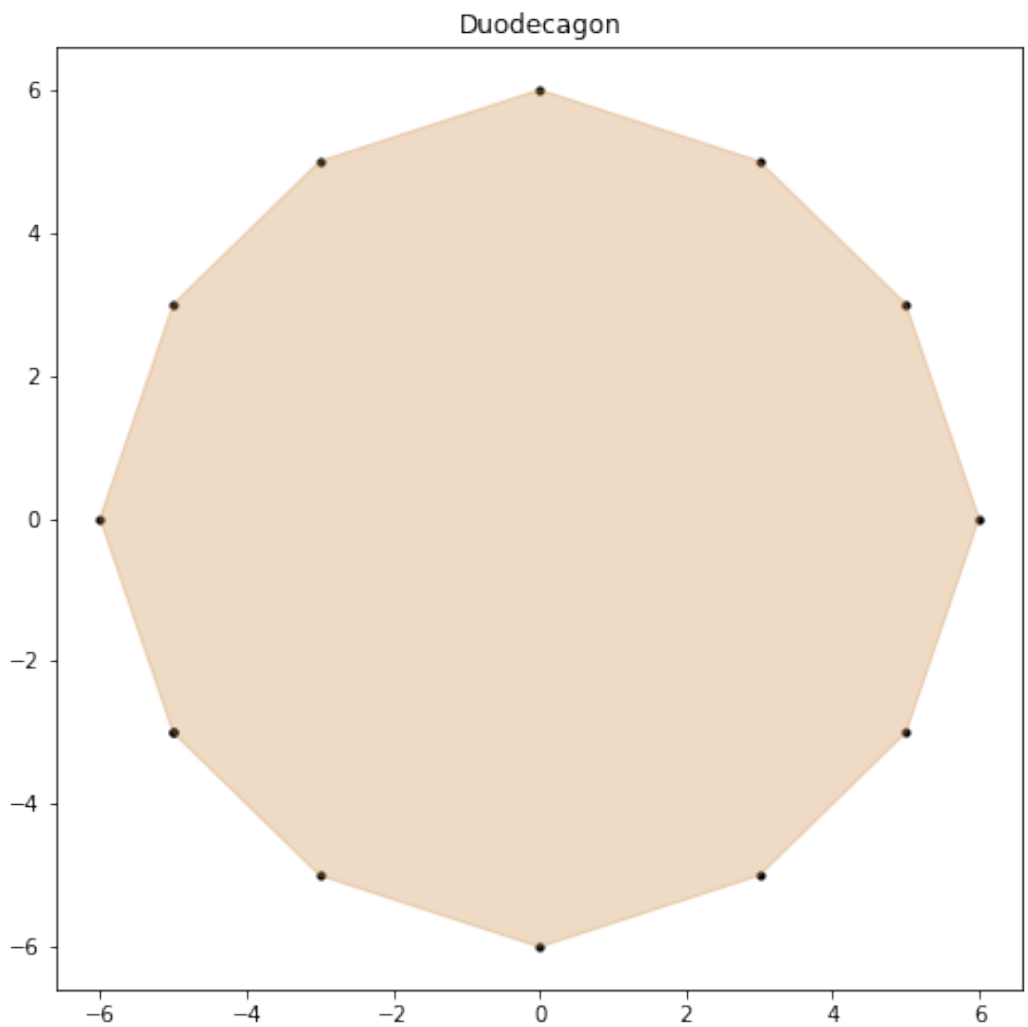
В качестве примера предлагается рассмотреть систему описывающую двенадцатиугольник:

```
>>> A = -np.array([[-3, -1],
>>>                [-2, -2],
>>>                [-1, -3],
>>>                [1, -3],
>>>                [2, -2],
>>>                [3, -1],
>>>                [3, 1],
>>>                [2, 2],
>>>                [1, 3],
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>>                [-1, 3],
>>>                [-2, 2],
>>>                [-3, 1]])
>>> b = -np.array([18,16,18,18,16,18,18,16,18,18,16,18])
>>> vertices = ip.lineqs(A, b, title='Duodecagon', color='peru', alpha=0.3, size=(8,8))
array([[-5., -3.], [-6., -0.], [-5.,  3.], [-3.,  5.], [-0.,  6.], [ 3.,  5.],
       [ 5.,  3.], [ 6.,  0.], [ 5., -3.], [ 3., -5.], [ 0., -6.], [-3., -5.]])
```



Трёхмерная визуализация линейной системы неравенств

Для работы с линейной системой алгебраических неравенств A x >= b, когда количество неизвестных равно трём, необходимо воспользоваться функций `lineqs3D`. В случае, если множество решений неограниченно, то алгоритм самостоятельно выберет границы отрисовки. Однако пользователь сам может указать их явным образом. Для понимания, что множество решений обрезано, плоскости окрашиваются в красный цвет.

Parameters:

- **A: float**
  Матрица системы линейных алгебраических неравенств.
- **b: float**
  Вектор правой части системы линейных алгебраических неравенств.
- **show: bool, optional**
  Данный параметр отвечает за то будет ли показано множество решений. По умолчанию указано значение True, т.е. происходит отрисовка графика.
- **color: str, optional**
  Цвет внутренней области множества решений.
- **bounds: array_like, optional**
  Границы отрисовочного окна. Первый элемент массива отвечает за нижние грани по осям OX, OY и OZ, а второй за верхние. Таким образом, для того, чтобы OX лежало в пределах [-2, 2], а OY в пределах [-3, 4], а OZ в пределах [1, 5] необходимо задать `bounds` как [[-2, -3, 1], [2, 4, 5]].
- **alpha: float, optional**
  Прозрачность графика.
- **s: float, optional**
  Насколько велики точки вершин.
- **size: tuple, optional**
  Размер отрисовочного окна.

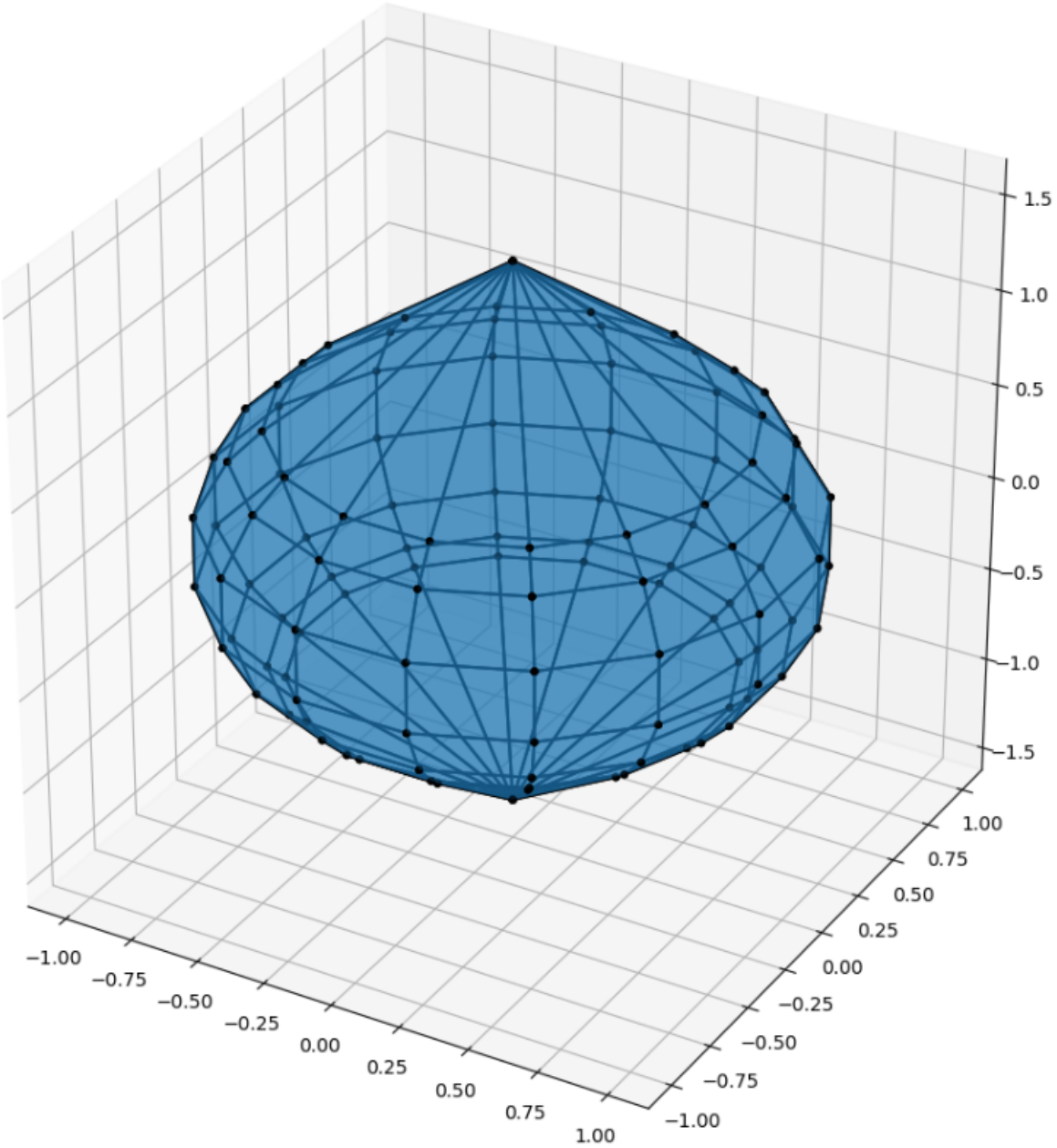
Returns:

- **out: list**
  Возвращается список упорядоченных вершин. В случае, если show = True, то график отрисовывается.

Examples:

В качестве примера предлагается рассмотреть систему описывающую юлу:

```
>>> %matplotlib notebook
>>> k = 4
>>> A = []
>>> for alpha in np.arange(0, 2*np.pi - 0.0001, np.pi/(2*k)):
>>>     for beta in np.arange(-np.pi/2, np.pi/2, np.pi/(2*k)):
>>>         Ai = -np.array([np.sin(alpha), np.cos(alpha), np.sin(beta)])
>>>         Ai /= np.sqrt(Ai @ Ai)
>>>         A.append(Ai)
>>> A = np.array(A)
>>> b = -np.ones(A.shape[0])
>>>
>>> vertices = ip.lineqs3D(A, b)
```



Визуализация множества решений ИСЛАУ с двумя неизвестными

Для работы с интервальной линейной системой алгебраических уравнений $\mathbf{A}\ \mathrm{x} = \mathbf{b}$, когда количество неизвестных равно двум, необходимо воспользоваться функций `IntLinIncR2`.

Для построения множества решений разобьём основную задачу на четыре подзадачи. Для этого воспользуемся свойством выпуклости решения в пересечении с каждым из ортантов пространства R^2, а также характеризацей Бекка. В результате получим задачи с системами линейных неравенств в каждом ортанте, которые можно визуализировать с помощью функции `lineqs`.

В случае, если множество решений неограниченно, то алгоритм самостоятельно выберет границы отрисовки. Однако пользователь сам может указать их явным образом.

Parameters:

- **A**
  [Interval] Входная интервальная матрица ИСЛАУ, которая может быть как квадратной, так и прямоугольной.
- **b**
  [Interval] Интервальной вектор правой части ИСЛАУ.
- **show: bool, optional**
  Данный параметр отвечает за то будет ли показано множество решений. По умолчанию указано значение True, т.е. происходит отрисовка графика.
- **title: str, optional**
  Верхняя легенда графика.
- **consistency: str, optional**
  Параметр для выбора типа множества решений. В случае, если он равен consistency = „uni“, то функция возвращает объединённое множество решение, если consistency = „tol“, то допусковое.
- **bounds: array_like, optional**
  Границы отрисовочного окна. Первый элемент массива отвечает за нижние грани по осям OX и OY, а второй за верхние. Таким образом, для того, чтобы OX лежало в пределах [-2, 2], а OY в пределах [-3, 4], необходимо задать `bounds` как [[-2, -3], [2, 4]].
- **color: str, optional**
  Цвет внутренней области множества решений.
- **alpha: float, optional**
  Прозрачность графика.
- **s: float, optional**
  Насколько велики точки вершин.
- **size: tuple, optional**
  Размер отрисовочного окна.
- **save: bool, optional**
  Если значение True, то график сохраняется.

Returns:

- **out: list**
  Возвращается список упорядоченных вершин в каждом ортанте начиная с первого и совершая обход в положительном направлении. В случае, если show = True, то график отрисовывается.

Examples:

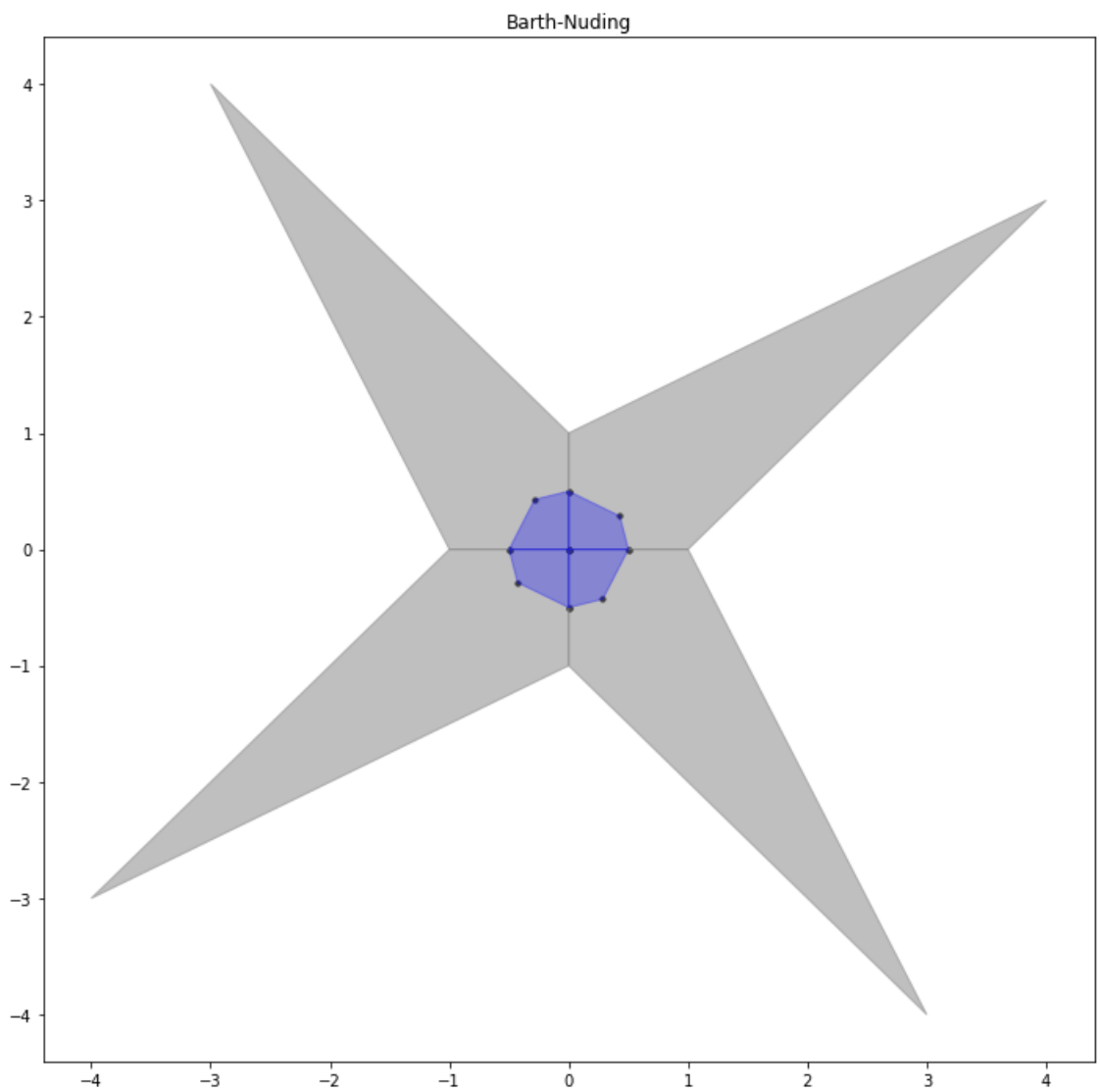
В качестве примера предлагается рассмотреть широкоизвестную интервальную систему предложенную Бартом-Нудингом. Для наглядности насколько отличаются разные типы решений изобразим на одном графике объединённое и допусковое множества:

```
>>> import matplotlib.pyplot as plt
>>>
>>> A = ip.Interval([[2, -2],[-1, 2]], [[4,1],[2,4]])
>>> b = ip.Interval([-2, -2], [2, 2])
>>>
>>> fig = plt.figure(figsize=(12,12))
>>> ax = fig.add_subplot(111, title='Barth-Nuding')
>>>
```

(continues on next page)

(продолжение с предыдущей страницы)

```
>>> vertices1 = ip.IntLinIncR2(A, b, show=False)
>>> vertices2 = ip.IntLinIncR2(A, b, consistency='tol', show=False)
>>>
>>> for v in vertices1:
>>>     # если пересечение с ортантом не пусто
>>>     if len(v) > 0:
>>>         x, y = v[:,0], v[:,1]
>>>         ax.fill(x, y, linestyle = '-', linewidth = 1, color='gray', alpha=0.5)
>>>         ax.scatter(x, y, s=0, color='black', alpha=1)
>>>
>>> for v in vertices2:
>>>     # если пересечение с ортантом не пусто
>>>     if len(v) > 0:
>>>         x, y = v[:,0], v[:,1]
>>>         ax.fill(x, y, linestyle = '-', linewidth = 1, color='blue', alpha=0.3)
>>>         ax.scatter(x, y, s=10, color='black', alpha=1)
```



Визуализация множества решений ИСЛАУ с тремя неизвестными

Для работы с интервальной линейной системой алгебраических уравнений $\mathbf{A}$ x = $\mathbf{b}$, когда количество неизвестных равно трём, необходимо воспользоваться функций `IntLinIncR3`.

Для построения множества решений разобьём основную задачу на восемь подзадач. Для этого воспользуемся свойством выпуклости решения в пересечении с каждым из ортантов пространства R^3, а также характеризацей Бекка. В результате получим задачи с системами линейных неравенств в каждом ортанте, которые можно визуализировать с помощью функции `lineqs3D`.

В случае, если множество решений неограниченно, то алгоритм самостоятельно выберет границы отрисовки. Однако пользователь сам может указать их явным образом. Для понимания, что множество решений обрезано, плоскости окрашиваются в красный цвет.

Parameters:

- **A**
  [Interval] Входная интервальная матрица ИСЛАУ, которая может быть как квадратной, так и прямоугольной.
- **b**
  [Interval] Интервальной вектор правой части ИСЛАУ.
- **show: bool, optional**
  Данный параметр отвечает за то будет ли показано множество решений. По умолчанию указано значение True, т.е. происходит отрисовка графика.
- **consistency: str, optional**
  Параметр для выбора типа множества решений. В случае, если он равен consistency = „uni“, то функция возвращает объединённое множество решение, если consistency = „tol“, то допусковое.
- **bounds: array_like, optional**
  Границы отрисовочного окна. Первый элемент массива отвечает за нижние грани по осям OX, OY и OZ, а второй за верхние. Таким образом, для того, чтобы OX лежало в пределах [-2, 2], а OY в пределах [-3, 4], а OZ в пределах [1, 5] необходимо задать `bounds` как [[-2, -3, 1], [2, 4, 5]].
- **color: str, optional**
  Цвет внутренней области множества решений.
- **alpha: float, optional**
  Прозрачность графика.
- **s: float, optional**
  Насколько велики точки вершин.
- **size: tuple, optional**
  Размер отрисовочного окна.

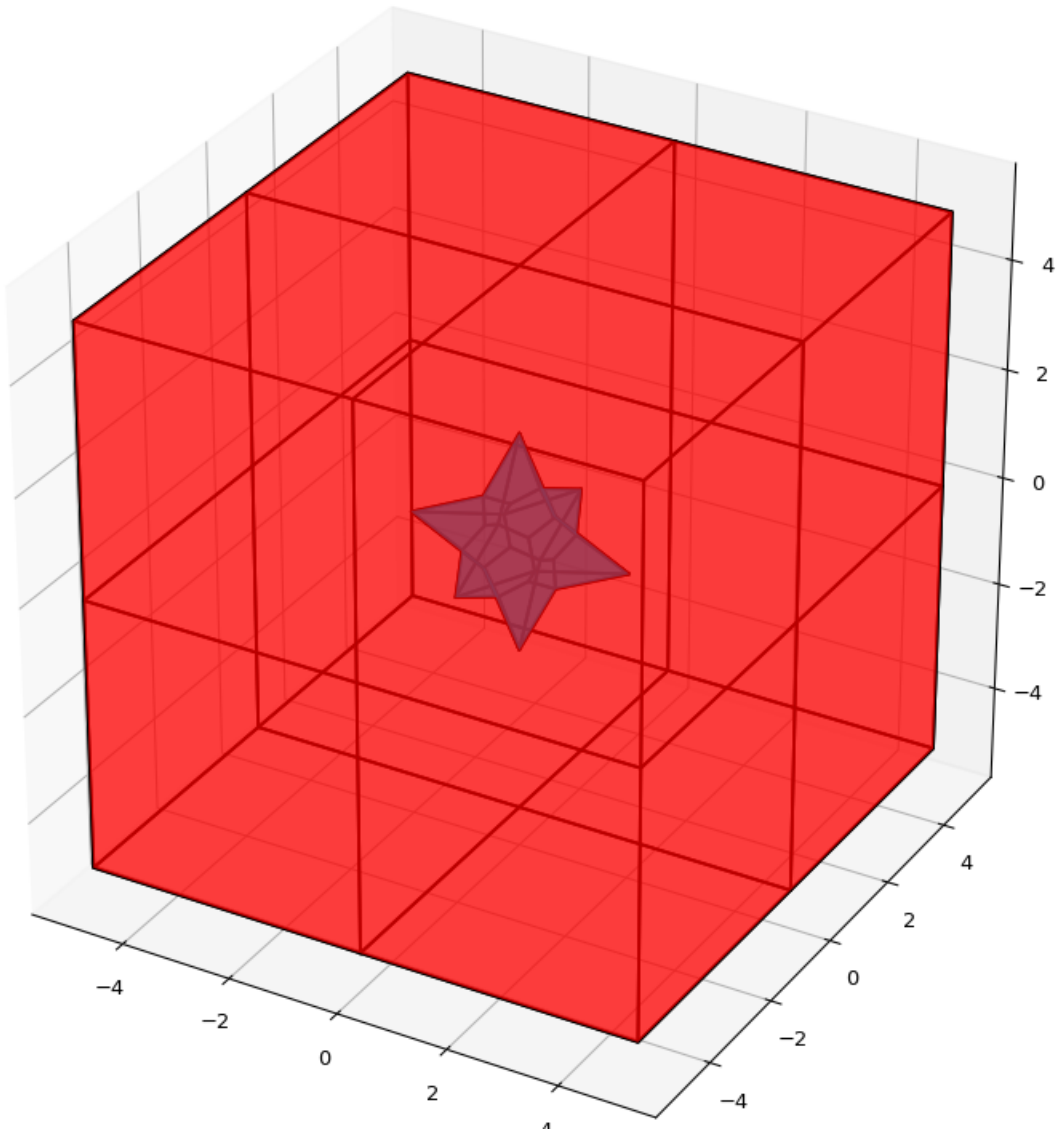
Returns:

- **out: list**
  Возвращается список упорядоченных вершин в каждом ортанте. В случае, если show = True, то график отрисовывается.

Examples:

В качестве примера рассмотрим интервальную систему у которой решением является вся область за исключением внутренности:

```
>>> %matplotlib notebook
>>> inf = np.array([[-1,-2,-2], [-2,-1,-2], [-2,-2,-1]])
>>> sup = np.array([[1,2,2], [2,1,2], [2,2,1]])
>>> A = ip.Interval(inf, sup)
>>> b = ip.Interval([2,2,2], [2,2,2])
>>>
>>> bounds = [[-5, -5, -5], [5, 5, 5]]
>>> vertices = ip.IntLinIncR3(A, b, alpha=0.5, s=0, bounds=bounds, size=(11,11))
```



Список использованной литературы

[1] И.А. Шарая - Метод граничных интервалов для визуализации полиэдральных множеств решений // Вычислительные технологии, Том 20, No 1, 2015, стр. 75-103.

[2] П.А. Щербина - Метод граничных интервалов в свободной системе компьютерной математики Scilab

[3] С.П. Шарый - Конечномерный интервальный анализ.

1.5.3 Методы для решения квадратных систем

В данном разделе предложены алгоритмы для решения квадратных интервальных систем уравнений.

Метод Гаусса

Метод исключения Гаусса, включая его различные модификации, крайне популярный алгортим в вычислительной линейной алгебре. Поэтому предлагается рассмотреть его интервальную версию, которая также состоит из двух этапов — *прямой ход* и *обратный ход*.

Parameters:

- **A**
  [Interval] Входная интервальная матрица ИСЛАУ, которая должна быть квадратной.
- **b**
  [Interval] Интервальной вектор правой части ИСЛАУ.

Returns:

- **out**
  [Interval] Интервальный вектор, который после подстановки в систему уравнений и выполнения всех операций по правилам арифметики и анализа обращает уравнения в инстинные равенства.

Examples:

В качестве примера рассмотрим широко известную интервальную систему, предложенную Бартом-Нудингом:

```
>>> A = ip.Interval([[2, -2],[-1, 2]], [[4, 1],[2, 4]])
>>> b = ip.Interval([-2, -2], [2, 2])
>>> ip.linear.Gauss(A, b)
interval(['[-5.0, 5.0]', '[-4.0, 4.0]'])
```

Interval Gauss-Seidel method

def Gauss_Seidel(A, b, x0=None, C=None, tol=1e-12, maxiter=2000)

The iterative Gauss-Seidel method for obtaining external evaluations of the united solution set for an interval system of linear algebraic equations (ISLAE).

Parameters:

- **A**
  [Interval] The input interval matrix of ISLAE, which can be either only square.
- **b**
  [Interval] The interval vector of the right part of the ISLAE.

- **X: Interval, optional**
  An initial guess within which to search for external evaluation is suggested. By default, X is an interval vector consisting of the elements [-1000, 1000].
- **C: np.array, Interval**
  A matrix for preconditioning the system. By default, C = inv(mid(A)).
- **tol: float, optional**
  The error that determines when further crushing of the bars is unnecessary, i.e. their width is «close enough» to zero, which can be considered exactly zero.
- **maxiter: int, optional**
  The maximum number of iterations.

Returns:

- **out**
  [Interval] Returns an interval vector, which means an external estimate of the united solution set.

Examples:

```
>>> A = ip.Interval([
        [[2, 4], [-2, 1]],
        [[-1, 2], [2, 4]]
    ])
>>> b = ip.Interval([[1, 2], [1, 2]])
>>> ip.linear.Gauss_Seidel(A, b)
Interval(['[-10.6623, 12.5714]', '[-11.0649, 12.4286]'])
```

Preconditioning the system with the inverse mean yields a vector of external evaluations that is wider than if a special type of preconditioning matrix were carefully selected in advance. The system presented below is the same system as described above, but preconditioned with a specially selected matrix.

```
>>> A = ip.Interval([[0.5, -0.456], [-0.438, 0.624]],
                    [[1.176, 0.448], [0.596, 1.36]])
>>> b = ip.Interval([0.316, 0.27], [0.632, 0.624])
>>> ip.linear.Gauss_Seidel(A, b, C=ip.eye(A.shape[0]))
Interval(['[-4.26676, 6.07681]', '[-5.37144, 5.26546]'])
```

Parameter partitioning methods

def PPS(A, b, tol=1e-12, maxiter=2000, nu=None)

PPS - optimal (exact) componentwise estimation of the united solution set to interval linear system of equations.

x = PPS(A, b) computes optimal componentwise lower and upper estimates of the solution set to interval linear system of equations Ax = b, where A - square interval matrix, b - interval right-hand side vector.

x = PPS(A, b, tol, maxiter, nu) computes vector x of optimal componentwise estimates of the solution set to interval linear system Ax = b with accuracy no more than epsilon and after the number of iterations no more than numit. Optional input argument ncomp indicates a component's number of interval solution in case of computing the estimates for this component only. If this argument is omitted, all componentwise estimates is computed.

Parameters:

- **A: Interval**
  The input interval matrix of ISLAE, which can be either square or rectangular.

- **b: Interval**
  The interval vector of the right part of the ISLAE.
- **tol: float, optional**
  The error that determines when further crushing of the bars is unnecessary, i.e. their width is «close enough» to zero, which can be considered exactly zero.
- **maxiter: int, optional**
  The maximum number of iterations.
- **nu: int, optional**
  Choosing the number of the component along which the set of solutions is evaluated.

Returns:

- **out: Interval**
  Returns an interval vector, which, after substituting into the system of equations and performing all operations according to the rules of arithmetic and analysis, turns the equations into true equalities.

Examples:

```
>>> A, b = ip.Neumeier(5, 10)
>>> ip.linear.PPS(A, b)
Interval(['[-0.214286, 0.214286]', '[-0.214286, 0.214286]', '[-0.214286, 0.214286]', '[-
↪0.214286, 0.214286]', '[-0.214286, 0.214286]'])
```

Список использованной литературы

[1] R.B. Kearfott, C. Hu, M. Novoa III - A review of preconditioners for the interval Gauss-Seidel method // Interval Computations, 1991-1, pp 59-85

[2] С.П. Шарый - Конечномерный интервальный анализ.

[3] S.P. Shary, D.Yu. Lyudvin - Testing Implementations of PPS-methods for Interval Linear Systems // Reliable Computing, 2013, Volume 19, pp 176-196

1.5.4 Методы для решения переопределённых систем

В случаях, когда рассматривается переопределённая интервальная система линейных алгебраических уравнений (ИСЛАУ), то если отбросить некоторые уравнения, чтобы привести систему к квадратному виду, то полученный вектор-решение будет содержать оптимальное оценивания множества решений. Однако такой приём может значительно ухудшить (раздуть) оценку, что, несомненно, является нежелательным. В связи с этим предлагается рассмотреть некоторые алгоритмы для решения переопределённых систем.

Метод Рона

Метод, предложенный Дж. Роном в статье [1], для получения вектора-решения, основан на решении вспомогательного квадратного линейного неравенства. Для получения данного неравенства активно используется наиболее представительная точечная матрица Ac из интеварльной матрицы $\mathbf{A}$, т.е. Ac = mid($\mathbf{A}$). Реализованный алгоритм является простейшей вариацией алгоритма предложенного в статье и *не* даёт оптимальное оценивание множества решений.

Parameters:

- **A**
  [Interval] Входная интервальная матрица ИСЛАУ, которая может быть как квадратной, так и прямоугольной.
- **b**
  [Interval] Интервальной вектор правой части ИСЛАУ.
- **tol**
  [float, optional] Погрешность, определающая, когда дальнейшее дробление брусов излишне, т.е. их ширина «достаточно близка» к нулю, что может считаться точно нулевой.
- **maxiter**
  [int, optional] Максимальное количество итераций для выполнения алгоритма.

Returns:

- **out**
  [Interval] Интервальный вектор, который после подстановки в систему уравнений и выполнения всех операций по правилам арифметики и анализа обращает уравнения в инстинные равенства.

Examples:

В качестве примера рассмотрим широко известную интервальную систему, предложенную Бартом-Нудингом:

```
>>> A = ip.Interval([[2, -2],[-1, 2]], [[4,1],[2,4]])
>>> b = ip.Interval([-2, -2], [2, 2])
>>> ip.linear.Rohn(A, b)
Interval(['[-14, 14]', '[-14, 14]'])
```

Этот пример также демонстрирует, что решение может быть далеко от оптимального, который в данном случае равен Interval([„[-4, 4]“, „[-4, 4]“]). В качестве второго примера предлагается рассмотреть тестовую систему С.П. Шарого:

```
>>> A, b = ip.Shary(4)
>>> ip.linear.Rohn(A, b)
Interval(['[-4.34783, 4.34783]', '[-4.34783, 4.34783]', '[-4.34783, 4.34783]', '[-4.
↪34783, 4.34783]'])
```

В отличие от прошлого примера данный вектор-решение достаточно близок к оптимальному внешнему оцениванию.

Метод дробления решений

Гибридный метод дробления решений PSS, подробно описанный в [2]. PSS-алгортимы предназначены для нахождения внешних оптимальных оценок множеств решений интервальных систем линейных алгебраических уравнений (ИСЛАУ) $\mathbf{A}$ x = $\mathbf{b}$.

В качестве базового метода внешнего оценивания в программе используется интервальный метод Гаусса (функция Gauss), если система является квадратной. В случае, если система переопределённая, то применяется простейший алгоритм, предложенный Дж. Роном (функция Rohn). Поскольку задача NP-трудная, то остановка процесса может произойти по количеству пройденных итераций. PSS-методы являются последовательно гарантирующими, т.е. при обрыве процесса на любом количестве итераций приближённая оценка решения удовлетворяет требуемому способу оценивания.

Возвращает формальное решение интервальной системы линейных уравнений. В случае, если оценивать все компоненты нет необходимости, то можно оценить одну любую nu-ю компоненту.

Parameters:

- **A**
  [Interval] Входная интервальная матрица ИСЛАУ, которая может быть как квадратной, так и прямоугольной.
- **b**
  [Interval] Интервальной вектор правой части ИСЛАУ.
- **tol**
  [float, optional] Погрешность, определающая, когда дальнейшее дробление брусов излишне, т.е. их ширина «достаточно близка» к нулю, что может считаться точно нулевой.
- **maxiter**
  [int, optional] Максимальное количество итераций для выполнения алгоритма.
- **nu**
  [int, optional] Выбор номера компоненты, вдоль которой оценивается множество решений.

Returns:

- **out**
  [Interval] Интервальный вектор, который после подстановки в систему уравнений и выполнения всех операций по правилам арифметики и анализа обращает уравнения в инстинные равенства.

Examples:

```
>>> A, b = ip.Shary(4)
>>> ip.linear.PSS(A, b)
interval(['[-4.347826, 4.347826]', '[-4.347826, 4.347826]', '[-4.347826, 4.347826]', '[-
↪4.347826, 4.347826]'])
```

Возврат интервального вектора решения NP-трудной системы.

```
>>> A, b = ip.Neumeier(3, 3.33)
>>> ip.linear.PSS(A, b, nu=0, maxiter=5000)
interval(['[-2.373013, 2.373013]'])
```

Возвращена отдельная компонента. В связи с тем, что в системе Ноймаера параметр theta=3.33 является жёстким условием, необходимо увеличить количество итераций для получения оптимальной оценки.

Список использованной литературы

[1] J. Rohn - Enclosing solutions of overdetermined systems of linear interval equations // Reliable Computing 2 (1996), 167-171

[2] С.П. Шарый - Конечномерный интервальный анализ.

[3] J. Horacek, M. Hladik - Computing enclosures of overdetermined interval linear systems // Reliable Computing 2 (2013), 142-155