TEMA Tend. Mat. Apl. Comput., **8**, No. 1 (2007), 73-82.

Módulo Python para Matemática Intervalar

P.S. GRIGOLETTI[1], PPGC, Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil

G.P. DIMURO[2], L.V. BARBOZA[3], PPGINF, Universidade Católica de Pelotas, Pelotas, RS, Brasil.

Resumo. Este trabalho apresenta o módulo *PyInterval* para Matemática Intervalar, implementado na linguagem Python e desenvolvido como software livre. Neste módulo foram disponibilizadas funcionalidades básicas, como, por exemplo, operações aritméticas com intervalos e matrizes de intervalos, funções intervalares (potência, exponencial, trigonométricas etc.), ponto médio, diâmetro, distância etc., além de alguns métodos para resolução de sistemas de equações lineares intervalares. Além de considerar o alto custo dos software proprietários (como o Matlab®, por exemplo) que, em geral, dificulta a ampla utilização de ferramentas para Matemática Intervalar, a opção pela linguagem Python baseou-se em sua simplicidade sintática, aliada à facilidade de implementação/extensão e portabilidade, e por ser multiplataforma e de livre distribuição. São apresentados exemplos de utilização e testes comparativos com resultados obtidos utilizando o *toolbox* IntLab.

1. Introdução

A modelagem computacional de eventos físicos apresenta limitações em termos de confiabilidade dos parâmetros e dados utilizados, que são obtidos, via de regra, por medições. Assim, estes valores apresentam intrinsecamente um erro, quer seja por falha humana (erro de leitura), quer seja por deficiência mecânica nos próprios instrumentos de medida (erro do instrumento) [13]. Por outro lado, a representação desses valores nos sistemas de ponto flutuante dos computadores digitais gera erros adicionais, pois estes sistemas nem sempre permitem que os números sejam armazenados com exatidão [14]. Surgem assim os erros de arredondamento e/ou truncamento, que também vão estar presentes na execução dos cálculos numéricos [13].

Assim, na modelagem computacional de eventos físicos, o tratamento da incerteza dos resultados aproximados obtidos é uma tarefa árdua. Para analisar o erro dos resultados destas computações, que, em alguns casos, são aproximações

[1]grigoletti@gmail.com. Este trabalho foi realizado quando P. S. Grigoletti estava vinculado à UCPel, Pelotas, RS

[2]liz@atlas.ucpel.tche.br

[3]luciano@atlas.ucpel.tche.br. Também atua no Centro Federal de Educação Tecnológica de Pelotas, Pelotas, RS

bastante pobres, é necessária a execução de tarefas auxiliares e/ou repetidas simulações. Observa-se que este tipo de análise, além de dispender um considerável esforço computacional, não é uma tarefa simples de ser operacionalizada.

A Matemática Intervalar é uma teoria matemática introduzida na década de 60 [15] que tem como objetivo fundamental o tratamento automático dos erros em Computação Científica [11]. Nesta abordagem, os valores incertos (pontuais) são armazenados através de intervalos, cujos extremos são pontos flutuantes. Estes intervalos encapsulam os dados incertos considerando o erro máximo que pode ter ocorrido[4]. Os erros (de arredondamento e/ou truncamento) que surgem posteriormente e que se propagam em todo processo computacional são tratados pela aritmética intervalar e pelos arredondamentos direcionados, que, pelo princípio da máxima exatidão, garantem o controle rigoroso dos erros nos resultados de computações numéricas [14]. Assim, ao final do processo matemático-computacional, tem-se uma estimativa da influência dos erros de entrada no resultado final obtido. Esta é uma análise automática, indicada pelo diâmetro do resultado intervalar final.

Existem diversas bibliotecas computacionais para Matemática Intervalar, como, por exemplo, o *toolbox* Intlab [17], desenvolvido para a ferramenta Matlab®. Este *toolbox* é de fácil utilização, apresenta vários recursos, além de rotinas para controlar o modo de arredondamento. No Intlab, as operações intervalares podem ser executadas sobre escalares intervalares reais e complexos, assim como vetores e matrizes, apresentando excelentes resultados na solução de SELA's intervalares. Embora este *toolbox* seja distribuído livremente, sua utilização é limitada pelo alto custo do software Matlab®. Encontrou-se esta mesma limitação no desenvolvimento da biblioteca MatInt [3], para a ferramenta Maple.

Por outro lado, a biblioteca C-XSC [12], um complexo conjunto de classes C++ para Computação Científica, consiste em uma ferramenta para o desenvolvimento de algoritmos numéricos para computação verificada e de alta exatidão. Esta biblioteca provê um grande número de tipos de dados numéricos e operadores pré-definidos, sendo um deles o tipo intervalo, o que permite a implementação de algoritmos intervalares verificados. Também salienta-se a existência da biblioteca para aritmética intervalar implementada em Sun Forte Developer C++ [19], mas, neste caso, agrega-se a dificuldade de implementação ao custo dos equipamentos, do sistema operacional Solaris e do compilador Forte.

Este trabalho apresenta o módulo *PyInterval* para Matemática Intervalar, desenvolvido na linguagem Python [18], baseado no paradigma de orientação a objetos e na concepção de software livre. O *PyInterval* é um conjunto de classes básicas para a manipulação de intervalos estendidos[5]. Além da preocupação com o custo dos softwares proprietários (como o Matlab®, por exemplo) que, em geral, dificulta a ampla utilização destas ferramentas, a opção pela linguagem Python baseou-se na sua simplicidade sintática, facilidade de implementação, portabilidade, características de multiplataforma e livre distribuição, permitindo assim que a comunidade científica possa criar extensões ao módulo, contribuindo assim para o seu desenvolvimento. Este módulo pode ser utilizado nas mais diversas áreas da Computação

[4]Os erros máximos em dados e parâmetros são fornecidos pelos especialistas, ou pelos fabricantes dos equipamentos e/ou componentes, que indicam as tolerâncias a serem consideradas.

[5]Intervalos estendidos são aqueles que admitem extremos infinitos [4].

Científica, e, portanto, além de possibilitar o desenvolvimento de aplicações intervalares, tem também como objetivo difundir a utilização da Matemática Intervalar.

Este artigo está organizado como descrito a seguir. Os principais conceitos da Matemática Intervalar são apresentados na Seção 2. A descrição do módulo *PyInterval* é introduzida na Seção 3. Exemplos comparativos com o *toolbox* Intlab são mostrados na Seção 4. As considerações finais são apresentadas na Seção 5.

2. Matemática Intervalar: Principais Conceitos

O *PyInterval* foi desenvolvido para manipular intervalos reais estendidos, uma abordagem não convencional, introduzida primeiramente em trabalhos de fundamentação da Matemática Intervalar [4], e utilizada pata tratamento de erros de *overflow*. Nesta abordagem, os números reais são estendidos com os elementos $-\infty$ e $+\infty$, tais que, para todo $x \in \mathbb{R}$, tem-se que: $-\infty < x$ e $x < +\infty$. Sobre o conjunto dos números reais, que denotaremos aqui por $\mathbb{R}^*$, desenvolveu-se uma aritmética e outros conceitos fundamentais (para estas definições, veja [4, 6]).

Um *intervalo real estendido* X é um conjunto não vazio $X = [x_1; x_2] = \{x \in \mathbb{R}^* \mid x_1 \leq x \leq x_2\}$, onde x_1 é o extremo inferior e x_2 é o extremo superior. Denota-se por $\mathbb{IR}^*$ a coleção de todos os intervalos reais estendidos. Para obter uma representação $\widetilde{X}$ de um intervalo real estendido $X = [x_1; x_2] \in \mathbb{IR}^*$ em um sistema de ponto flutuante de uma máquina, os elementos $-\infty$ e $+\infty$ são interpretados como o menor e o maior números de ponto flutuante da máquina, respectivamente, e os extremos x_1 e x_2 devem ser arredondados "por falta" e "por excesso", respectivamente, o que denomina-se *arredondamento direcionado*. Tem-se então que $X \subseteq \widetilde{X}$.

O *ponto médio*, o *diâmetro* e o *raio* de um intervalo X são definidos, respectivamente, como $mid(X) = \breve{X} = \frac{1}{2}(x_1 + x_2)$, $diam(X) = x_2 - x_1$ e $rad(X) = \frac{1}{2} diam(X)$.

As *operações aritméticas intervalares* são definidas como $X * Y = \{x * y \mid x \in X, y \in Y\}$, para $* \in \{+, -, \times, \div\}$. Se $X = [x_1; x_2]$ e $Y = [y_1; y_2]$, tem-se que [15]:

$$\begin{aligned} X + Y &= [x_1 + y_1; x_2 + y_2], \quad X - Y = [x_1 - y_2; x_2 - y_1] \\ X \times Y &= [\min \rho; \max \rho], \text{ com } \rho = \{x_1 y_1, x_1 y_2, x_2 y_1, x_2 y_2\} \\ X \div Y &= X \times [y_2^{-1}; y_1^{-1}]. \end{aligned}$$

Observa-se que a adição e multiplicação são associativas e comutativas. Entretanto, a propriedade da distributividade nem sempre é válida, causando superestimativa [15]. As operações aritméticas são monotônicas para a inclusão.

A *imagem* de uma função real $f : \mathbb{R}^* \to \mathbb{R}^*$ de um intervalo $X \in \mathbb{IR}^*$ é dada por $\bar{f}(X) = \{f(x) \mid x \in X\}$. A *extensão intervalar* de uma função real f é uma função intervalar $F(X) : \mathbb{IR}^* \to \mathbb{IR}^*$ com a propriedade de que $F(x) = f(x)$, se $x = [x; x] \in \mathbb{R}^*$. Não existe uma extensão intervalar única para uma dada função real, mas sempre tem-se que $\bar{f}(X) \subseteq F(X)$ [15].

Diversos métodos para o cálculo de inclusões de imagens de funções reais foram desenvolvidos. Também, vários métodos da Análise Numérica receberam versões intervalares, como é o caso do clássico Método de Newton Intervalar e suas extensões [1], para a solução de equações. Mas também tem sido extensiva a pesquisa de novos algoritmos intervalares para a solução de problemas relacionados

às mais diversas áreas onde o tratamento da incerteza é fundamental [11], como, por exemplo, Inteligência Artificial e Sistemas Multiagentes [5], *Soft Computing*, Geoinformática, Engenharia Elétrica [7] etc., oferecendo métodos para sistemas de equações lineares e não lineares, otimização, equações diferenciais etc. Em particular, os métodos para sistemas lineares intervalares têm sido um constante foco de pesquisa; foram introduzidos por E. Hansen [8] e continuados por um grande número de pesquisadores [9, 10, 16], inclusive em abordagens de programação paralela [2].

3. O Módulo *PyInterval*

O módulo *PyInterval* contempla os conceitos básicos da Matemática Intervalar, como, por exemplo, a aritmética e outras operações, o cálculo de funções (potência, exponencial, trigonométricas etc.), definições complementares (ponto médio, valor absoluto, diâmetro etc.), operações com matrizes intervalares e solução de SELA's intervalares. Foram implementados os arredondamentos direcionados, o que garante que o resultado exato sempre estará contido no intervalo resultante.

O módulo *PyInterval* é licenciado pela GPL, ou seja, tem seu código aberto e é distribuído como software livre. Deste modo, o *PyInterval* foi desenvolvido de forma a propiciar, se necessário, a expansão do mesmo, conforme as necessidades de cada usuário. Além disso, o módulo é portável e multiplataforma, características estas herdadas da linguagem de programação Python.

3.1. A Arquitetura

O módulo *PyInterval* foi desenvolvido segundo o paradigma de orientação a objetos, sendo composto basicamente pelas seguintes classes: (*i*) Globals, (*ii*) EFloat, (*iii*) Interval, (*iv*) IMatrix e (*v*) IEquationSolve, conforme o diagrama da Figura 1.

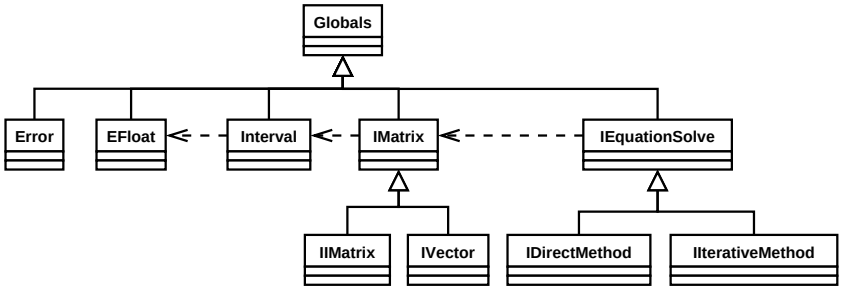

Figura 1: Diagrama de classes do módulo *PyInterval*.

A classe *Globals* é utilizada na definição de variáveis globais. Todas as outras classes herdam desta classe, que define, por exemplo, o maior e o menor número de máquina. A classe *EFloat* trabalha com números de ponto flutuante estendidos, que fornecem suporte aos elementos $-\infty$ e $+\infty$, que representam o maior e o menor número representável na máquina e são utilizados como uma alternativa para o tratamento dos erros de *overflow*. A classe *Interval* possibilita o trabalho com

intervalos que utilizam, como extremos, números de ponto flutuante estendidos, suportando desde intervalos muito pequenos até intervalos como $[-\infty; +\infty]$. Assim, cada extremo do intervalo é um objeto da classe *EFloat*. A classe *EFloat* pode ser utilizada de forma independente da classe *Interval*, possibilitando operações lógicas e aritméticas com números de ponto flutuante estendidos.

A classe *IMatrix* trabalha com matrizes intervalares (matrizes cujos elementos são objetos da classe *Interval*). Duas classes herdam desta: (*i*) a classe *IIMatrix*, para matrizes identidades intervalares e (*ii*) a classe *IVector*, para vetores intervalares. A classe *IEquationSolve* fornece a possibilidade da resolução de SELA's intervalares. Um SELA intervalar é representado por um sistema do tipo $Ax = b$, onde A é uma matriz intervalar (objeto *IMatrix*), b e x são vetores intervalares (objetos *IVector*). Duas classes herdam desta: (*i*) a *IDirectMethod*, que implementa métodos diretos e a (*ii*) *IIterativeMethod*, que implementa métodos iterativos.

3.2. Exemplos de Utilização

Esta seção apresenta alguns exemplos de utilização do módulo *PyInterval*. Nos exemplos a seguir, primeiramente, é necessário fazer a importação do módulo. O comando *print* imprime o valor da variável.

A Figura 2 mostra exemplos de operações com números de ponto flutuante estendidos e o arredondamento direcionado. São definidos dois números de ponto flutuante estendidos (`ef1, ef2`) e uma terceira variável (`ef3`) recebe o resultado da soma das duas primeiras (`ef1, ef2`), que, posteriormente é arredondada "por excesso". Após, é introduzido o maior número possível (`ef4`) e uma quinta variável (`ef5`) recebe uma soma envolvendo este valor, o que geraria, em geral, um erro de *overflow*, evitado aqui pela aritmética estendida.

Na Figura 3, apresenta-se um exemplo de operação com intervalos. São definidos dois intervalos (`i1, i2`) e uma terceira variável (`i3`) recebe o resultado da soma dos dois primeiros intervalos.

A Figura 4 traz um exemplo de resolução de um SELA intervalar. São definidos uma matriz (`m1`) e um vetor intervalar (`v1`), que são utilizados na resolução de um SELA intervalar utilizando um método iterativo, onde x é o vetor resultante.

A Figura 5 mostra um exemplo de operação com matrizes intervalares, onde definem-se duas matrizes intervalares (`m1, m2`) e uma terceira variável (`m3`) recebe o resultado da soma das duas primeiras matrizes.

4. Testes Comparativos

Nesta seção são apresentadas algumas comparações entre os resultados obtidos utilizando o módulo *PyInterval* e o IntLab [17] (os testes completos estão em [6]). Sendo este último uma referência na comunidade intervalar, nosso objetivo é mostrar que os resultados obtidos pelo *PyInterval* são aceitáveis e de boa qualidade.

As Tabelas 1 e 2 mostram os resultados obtidos no cálculo de operações aritméticas intervalares, utilizando o *PyInterval* e o IntLab, respectivamente. É possível observar que, no momento em que a atribuição dos intervalos é realizada, os intervalos no *PyInterval* possuem maior diâmetro. Isto se deve ao arredondamento

```
>>> from PyInterval import *
>>> ef1 = Efloat(10)
>>> ef2 = Efloat(7)
>>> ef3 = ef1 / ef2
>>> print ef3
1.4285714285714286
>>> ef3.roundUp()
>>> print ef3
1.4285714285714288
>>> ef4 =
efloat(1.6999999999999999e+308)
>>> ef5 = ef4 * ef1
>>> print ef5
+oo
```

Figura 2: Operações com números de ponto flutuante estendidos.

```
>>> from PyInterval import *
>>> i1 = Interval(1,2)
>>> i2 = Interval(-1,2)
>>> print i1, i2
[1.0;2.0] [-1.0;2.0]
>>> i3 = i1 + i2
>>> print i3
[0.0;4.0]
>>> i4 = i1 / i2
>>> print i4
[-oo,+oo]
>>> i5 = i4 + i1
>>> print i5
[-oo,+oo]
```

Figura 3: Operação com intervalos.

```
>>> from PyInterval import *
>>> m1 = IMatrix([[1,2],[4,3]])
>>> v1 = IVector([5,10])
>>> s = IIterativeMethod(m1,v1)
>>> print s.getXVector()
[[0.99999999999999023;
  1.0000000000000104]
 [1.9999999999999916;
  2.0000000000000089]]
```

Figura 4: Resolução de um SELA intervalar.

```
>>> from PyInterval import *
>>> m1 = IMatrix([[1,2],[3,4]])
>>> m2 = IMatrix([[[1.1,1.2],[2.1,2.2]], [[3.1,3.2],[4.1,4.2]]])
>>> print m1
[[[1.0;1.0] [2.0;2.0] ] [[3.0;3.0] [4.0;4.0] ]]
>>> print m2
[[[1.0999999999999999;1.2000000000000002] [2.0999999999999996;2.2000000000000006] ]
 [[3.0999999999999996;3.2000000000000006] [4.0999999999999988;4.2000000000000011] ]]
>>> m3 = m1+m2
>>> print m3
[[[2.0999999999999988;2.2000000000000011] [4.0999999999999979;4.2000000000000028] ]
 [[6.0999999999999979;6.2000000000000028] [8.0999999999999943;8.2000000000000046] ]]
```

Figura 5: Operação com matrizes intervalares.

direcionado aplicado diretamente aos dados de entrada. Com relação aos resultados, o *PyInterval* obteve intervalos de menor diâmetro na soma e na divisão.

Tabela 1: Operações aritméticas intervalares utilizando o módulo *PyInterval*.

Operações	**Resultados no *PyInterval***
intervalo1 = [1.1;1.2]	[1.0999999999999999;1.2000000000000002]
intervalo2 = [3.2;3.4]	[3.1999999999999997;3.4000000000000004]
intervalo1+intervalo2 =	[4.2999999999999989;4.6000000000000014]
intervalo1−intervalo2 =	[-2.3000000000000012;-1.9999999999999993]
intervalo1×intervalo2 =	[3.5199999999999987;4.0800000000000018]
intervalo1/intervalo2 =	[0.32352941176470568;0.37500000000000017]

As Tabelas 3 e 4 apresentam os resultados de operações matriciais intervalares, obtidos com o *PyInterval* e o IntLab, respectivamente. Os resultados apresentados pelas duas ferramentas são muito semelhantes. O módulo *PyInterval*, em certos casos, gerou intervalos de menor diâmetro devido à sua forma de representar os números, com um maior número de casas decimais.

A Tabela 5 mostra a matriz A e o vetor b^T de um SELA intervalar (na forma $Ax = b$). As soluções obtidas (o vetor x) são apresentadas na Tabela 6.

Tabela 2: Operações aritméticas intervalares utilizando o *toolbox* IntLab.

Operações	**Resultados no *toolbox* IntLab**
intervalo1 = [1.1;1.2]	`[ 1.10000000000000, 1.20000000000000]`
intervalo2 = [3.2;3.4]	`[ 3.20000000000000, 3.40000000000000]`
intervalo1+intervalo2 =	`[ 4.29999999999998, 4.60000000000001]`
intervalo1−intervalo2 =	`[ -2.30000000000000, -2.00000000000000]`
intervalo1×intervalo2 =	`[ 3.52000000000000, 4.08000000000001]`
intervalo1/intervalo2 =	`[ 0.32352941176470, 0.37500000000001]`

Tabela 3: Operações matriciais intervalares no módulo *PyInterval*.

Definição das matrizes intervalares =

```
>>> matriz1 = IMatrix([[[1.1,1.2],[-1.8,-1.7]],[[9.1,9.2],[8.7,8.8]]])
>>> matriz2 = IMatrix([[[6.1,6.2],[-3.8,-3.7]],[[8.1,8.2],[1.7,1.8]]])
```

matriz1 =

```
 [[[1.0999999999999999;1.2000000000000002] [-1.8000000000000003;-1.6999999999999997]]
  [[9.0999999999999979;9.2000000000000011] [8.6999999999999975;8.8000000000000025]]]
```

matriz2 =

```
 [[[6.0999999999999988;6.2000000000000011] [-3.8000000000000003;-3.6999999999999997]]
  [[8.0999999999999979;8.2000000000000011] [1.6999999999999997;1.8000000000000003]]]
```

matriz1+matriz2 =

```
 [[[7.1999999999999966;7.400000000000003] [-5.6000000000000023;-5.3999999999999977]]
  [[17.199999999999989;17.400000000000009] [10.399999999999993;10.600000000000007]]]
```

matriz1−matriz2 =

```
 [[[-5.1000000000000032;-4.8999999999999968] [1.899999999999999;2.1000000000000014]]
  [[0.89999999999999658;1.1000000000000036] [6.899999999999995;7.100000000000005]]]
```

matriz1×matriz2 =

```
 [[[-8.0500000000000114;-6.3299999999999876] [-7.800000000000006;-6.9599999999999955]]
  [[125.97999999999988;129.2000000000001] [-20.17000000000003;-17.829999999999963]]]
```

5. Considerações Finais

Este trabalho apresentou o módulo *PyInterval* para Matemática Intervalar, implementado na linguagem Python, sendo uma opção em software livre para o desenvolvimento de aplicações com controle automático e rigoroso de resultados. Este módulo disponibiliza apenas as funções essenciais para a utilização de técnicas intervalares, podendo ser facilmente estendido e incorporado a outras aplicações. Em [6, 7], por exemplo, este módulo foi a base do desenvolvimento da ferramenta FINCA (*Free Interval Circuit Analyzer*) para a análise intervalar de circuitos elétricos, hoje em utilização no curso de Engenharia Elétrica da UCPel e em cursos técnicos do SENAI/RS. Também foram desenvolvidas extensões do módulo para trabalhar com cadeias de Markov e modelos ocultos de Markov intervalares, que estão sendo utilizados para reconhecimento de comportamento de agentes em sistemas abertos [5].

Foram realizados testes e comparações com o IntLab, comprovando que os resultados fornecidos pelo *PyInterval* são de boa qualidade e confiáveis. Utilizando uma

Tabela 4: Operações matriciais intervalares no *toolbox* IntLab.

Definição das matrizes intervalares =
`>> matriz1 = [infsup(1.1,1.2),infsup(-1.8,-1.7);infsup(9.1,9.2),infsup(8.7,8.8)]`
`>> matriz2 = [infsup(6.1,6.2),infsup(-3.8,-3.7);infsup(8.1,8.2),infsup(1.7,1.8)]`
matriz1 =
`[ 1.10000000000000, 1.20000000000000] [ -1.80000000000001, -1.69999999999999]`
`[ 9.09999999999998, 9.20000000000001] [ 8.69999999999999, 8.80000000000001]`
matriz2 =
`[ 6.09999999999998, 6.20000000000001] [ -3.80000000000000, -3.70000000000000]`
`[ 8.09999999999998, 8.20000000000001] [ 1.69999999999999, 1.80000000000001]`
matriz1+matriz2 =
`[ 7.19999999999999, 7.40000000000001] [ -5.60000000000001, -5.39999999999998]`
`[ 17.19999999999999, 17.40000000000000] [ 10.39999999999999, 10.60000000000001]`
matriz1−matriz2 =
`[ -5.10000000000001, -4.89999999999998] [ 1.90000000000000, 2.10000000000001]`
`[ 0.90000000000000, 1.10000000000000] [ 6.89999999999999, 7.10000000000001]`
matriz1×matriz2 = `1.0e+002 *`
`[ -0.08055000000001, -0.06324999999999] [ -0.07800000000001, -0.06949999999999]`
`[ 1.25969999999999, 1.29200000000001] [ -0.20175000000001, -0.17824999999998]`

precisão maior, o *PyInterval*, em alguns casos, apresentou inclusive resultados com menor diâmetro. Entretanto, ainda não foi possível disponibilizar todas as funcionalidades oferecidas pelo IntLab, como, por exemplo, vários métodos para resolução de SELA's intervalares. Os métodos implementados no *PyInterval* apenas focaram a aplicação pretendida (análise intervalar de circuitos elétricos), sendo escolhidos os que melhor se comportaram para aquele tipo de aplicação (veja em [7]).

Considerando o custo de software proprietários, o *PyInterval* é uma boa alternativa para o desenvolvimento de aplicações intervalares. A simplicidade sintática da linguagem Python facilita a implementação de extensões pela comunidade científica, que tem contribuído para o contínuo desenvolvimento do módulo *PyInterval*. A sua utilização é garantida pelas características de portabilidade e multiplataforma.

Agradecimentos.
Este trabalho foi parcialmente financiado por CNPq e FAPERGS. Agradecemos aos revisores pelos comentários e sugestões recebidas.

Abstract. This works presents the *PyInterval* module for Interval Mathematics, implemented in Python. Basic functionalities are available, such as: interval arithmetic operations and interval matrix operations, interval functions, mid point, diameter, distance etc., and some methods for the solution of interval linear systems. Considering the cost of property software (like, e.g., Matlab®), which, in general, makes difficult the usage of such tools, we chose Python mainly for it's syntactic simplicity, portability, multiplatform and free distribution features, which allow the development of extensions by the scientific community. The paper presents examples and comparative tests with IntLab.

Tabela 5: Dados de um SELA intervalar na forma $Ax = b$.

Matriz $A =$	`[[[1.0;1.0] [0.0;0.0] [0.0;0.0] [0.0;0.0] [0.0;0.0] [0.0;0.0]]`
	`[[1.0;1.0] [-1.0;-1.0] [0.0;0.0] [0.0;0.0] [0.0;0.0] [0.0;0.0]]`
	`[[0.0;0.0] [1.0;1.0] [-1.0;-1.0] [0.0;0.0] [0.0;0.0] [0.0;0.0]]`
	`[[0.0;0.0] [0.0;0.0] [-0.2777777777777812;-0.22727272727272704]`
	`[0.45454545454545403;0.55555555555555625]`
	`[0.22727272727272704;0.27777777777777812]`
	`[0.22727272727272704;0.27777777777777812]]`
	`[[0.0;0.0] [0.0;0.0] [0.0;0.0] [1.0;1.0] [-1.0;-1.0] [0.0;0.0]]`
	`[[0.0;0.0] [0.0;0.0] [0.0;0.0] [0.0;0.0] [1.0;1.0] [-1.0;-1.0]]]`

Vetor $b^T =$	`[[10.0;10.0] [8.0;8.0] [6.0;6.0] [0.0;0.0] [4.0;4.0] [3.0;3.0]]`

Tabela 6: Resultados da solução do SELA intervalar.

PyInterval	IntLab
Vetor $x=$	**Vetor** $x=$
`[[10.0;10.0]`	`[ 10.00000000000000, 10.00000000000000]`
`[2.0;2.0]`	`[ 2.00000000000000, 2.00000000000000]`
`[-4.0;-4.0]`	`[ -4.00000000000000, -4.00000000000000]`
`[1.3333333333333037;2.1666666666666963]`	`[ 1.32962499999999, 2.17037500000001]`
`[-2.6666666666666963;-1.8333333333333028]`	`[ -2.67037500000001, -1.82962499999999]`
`[-5.6666666666667016;-4.8333333333332966]]`	`[ -5.67037500000001, -4.82962499999999]`

Referências

[1] G. Alefeld, J. Herzberger, "Introduction to Interval Computations", Academic Press, New York, 1983.

[2] T. Beelitz, B. Lang, C.H. Bischof, Efficient Task Scheduling in the Parallel Result-Verifying Solution of Nonlinear Systems, *Reliable Computing*, **12**, No. 2 (2006), 141–151.

[3] A.M. Dias, G.P. Dimuro, "Matemática Intervalar com Aplicações no Maple", NAPI/UCPel, Pelotas, 2000. (disponível em http://gmc.ucpel.tche.br/mat-int/)

[4] G.P. Dimuro, "Domínios Intervalares da Matemática Computacional", PPGC/UFRGS, Porto Alegre, 1991.

[5] G.P. Dimuro, A.C.R. Costa, L.V. Gonçalves, A. Hübner, L.V. Barboza, M.A. Campos, V. Kreinovich, Interval-valued Hidden Markov Models for recognizing personality traits in social exchanges in open multiagent systems, in "Proc. 12th GAMM-IMACS Intl. Symp. Scientific Computing, Computer Arithmetic and Validated Numerics", pp. 148–159, Univ. Duisburg-Essen, 2006.

[6] P.S. Grigoletti, "Uma Ferramenta Computacional Intervalar para a Análise Confiável de Circuitos Elétricos", ESIN/UCPel, Pelotas, 2004.

[7] P.S. Grigoletti, G.P. Dimuro, L.V. Barboza, R.H.S. Reiser, Análise intervalar de circuitos elétricos, em "Seleta do XXVIII CNMAC" (C.F. Bracciali, V.L.R.

Lopes, A.J. Silva Neto, A. Sri Ranga, H.M. Yang, eds.), TEMA - Tendências em Matemática Aplicada e Computacional, Vol. 7, No. 2, pp. 287–296, SBMAC, 2006.

[8] E.R. Hansen, On solving systems of equations using interval arithmetic, *Mathematics of Computation*, **22**, No. 102 (1968), 374–384.

[9] E.R. Hansen, Bounding the solution of interval linear equations, *SIAM Journal on Numeric Analysis*, **29**, No. 5 (1992), 1493–1503.

[10] E.R. Hansen, G.W. Walster, Solving overdetermined systems of interval linear equations, *Reliable Computing*, **12**, No. 3 (2006), 239–243.

[11] R.B. Keafort, V. Kreinovich (eds.), "Applications of Interval Computations", Kluwer, Boston, 1996.

[12] R. Klatte, U. Kulisch, C. Lawo, M. Rauch, A. Wietho, "C-XSC – A C++ class library for extended scientific computing", Springer, Heidelberg, 1993.

[13] U. Kulisch, W.L. Miranker (eds.), "A New Approach to Scientific Computation", Academic Press, New York, 1983.

[14] U. Kulisch, Advanced arithmetic for the digital computer, design of arithmetic units, *Electronic Notes in Theoretical Computer Science*, **24** (1999), 63.

[15] R.E. Moore, "Methods and Applications of Interval Analysis", SIAM, Philadelphia, 1979.

[16] A. Neumaier, "Interval Methods for Systems of Equations", Cambridge University Press, Cambridge, 1990.

[17] S.M. Rump, IntLab - Interval Laboratory, in "Developments in Reliable Computing" (T. Csendes, ed.), Kluwer, Dordrecht, 1999.

[18] G. Van Rossum, "Python tutorial: release 2.4.3", Python Software Foundation, 2006. (disponível em http://www.python.org/doc/current/tut/)

[19] Sun Microsystems, "Sun Forte Developer 7: C++ Interval Arithmetic Programming Reference", Santa Clara, 2002. (disponível em http://docs.sun.com/)