Linguagem de programação JULIA: uma alternativa *open source* e de alto desempenho ao MATLAB

João Marcello Pereira [1], Mario Benjamim Baptista de Siqueira [2]

[1] jmarcellopereira@ufpi.edu.br. Universidade Federal do Piauí - Bom Jesus, PI. [2] mariosiqueira@unb.br. Universidade de Brasília - Brasília, Distrito Federal.

RESUMO

O MATLAB® é um dos principais softwares utilizados nos cursos de ciências exatas e de engenharia para o ensino de programação numérica e pesquisa científica, em função de sua sintaxe de alto nível e dos diversos toolboxes do seu amigável ambiente de programação. Apesar das muitas vantagens, é um software proprietário que apresenta um alto custo de aquisição para as instituições de ensino e uso individual. Além disso, possui código fonte fechado e os programas nativos ".m" não são plenamente compatíveis em outros ambientes de programação com suporte a esse formato de arquivo. Embora existam softwares livres e open source com relativa semelhança de IDE (Integrated Development Environment ou Ambiente de Desenvolvimento Integrado) e sintaxe de código, muitos desses softwares apresentam baixo desempenho computacional em relação ao MATLAB®. Dessa forma, o objetivo no presente trabalho é apresentar a linguagem de programação Julia, como alternativa ao MATLAB®, no ensino de programação numérica e simbólica, bem como de pesquisa científica. Neste trabalho são comparadas as características das duas linguagens de programação, sendo apresentado um *benchmark* entre as linguagens em que se avaliam o tempo de execução e os resultados dos cálculos. Para isso, foram implementados em JULIA e em MATLAB® os algoritmos Série de Fibonacci Recursiva e Gráfico 2D dos polinômios de Berstein e, ainda, utilizadas as funções nativas de cálculo simbólico, algébrico e equação diferencial numérica Runge-Kutta 45.

Palavras-chave: MatLab, Julia, Linguagens de Programação, *Benchmark*, Programação.

ABSTRACT

The MATLAB® is one of the main software used in exact sciences and engineering courses for the teaching of programming and numerical scientific research, because of its high level syntax and several toolboxes from your friendly programming environment. Despite the many advantages, it is proprietary software that presents the high cost of licensing for Educational Institutions and individual use. Besides, it has closed source code and native programs ".m" is not fully compatible in other programming environments that support this file format. Although there are free software's and open-source projects with relative similarity of IDE and code syntax, many of these have lower computing performance when compared to MATLAB®. Thus, the present study aims to present the programming language JULIA as an alternative to MATLAB® in numerical and symbolic programming education, and scientific research. In this work the characteristics of the two programming languages are compared and benchmark computations between the languages are presented in which it is evaluated the runtime and the results of the calculations. For this, it has been implemented the Recursive Fibonacci series algorithms, Berstein polynomials 2D plots, native functions of symbolic calculus, algebraic operations and Runge-Kutta 45 solution for numerical differential equation. JULIA has shown competitive performance compared to MATLAB® in the tests made, being an interesting alternative in engineering education and research.

Keywords: *MatLab, Julia, Programming Languages, Benchmark, Programming.*

1 Introdução

A análise numérica permite que inúmeros problemas matemáticos, cujas soluções algébricas são difíceis, ou até mesmo impossíveis de se encontrar, possam ser resolvidos por meio de métodos numéricos apropriados (SPERANDIO; MENDES; SILVA, 2003).

Dessa forma, diversos softwares e linguagens de programação foram desenvolvidos com o intuito de resolver problemas numéricos, sendo aplicados, principalmente, em pesquisas e no ensino das disciplinas de cálculo e métodos numéricos. Entre esses, o MATLAB® é a mais expressiva plataforma (linguagem e ambiente de programação) de programação numérica e simbólica utilizada nas instituições de ensino e nas indústrias. Embora seja muito popular no ambiente acadêmico, o MATLAB® é uma plataforma fechada de alto custo de aquisição, além de exigir computadores acima do padrão básico do mercado brasileiro.

Realizadas as considerações, é oportuno destacar que, a linguagem de programação Julia foi lançada em 2012 sob a licença open source do MIT (*Massachusetts Institute of Technology*), com uma proposta ousada de substituir todas as linguagens de computação científica.

A linguagem Julia apresenta sintaxe de comandos semelhante ao MATLAB®, liberdade de escolha da IDE, não exigindo máquinas robustas, podendo ser utilizada livremente por meio do site juliabox.com, o qual conta com a infraestrutura da Amazon.com. No site oficial da linguagem, os desenvolvedores publicaram um *benchmark* entre outras linguagens nos quais os dados publicados demonstram um desempenho significativo. Os resultados podem ser visto na Tabela 1.

Tabela 1 – Tempo de execução em relação ao tempo de C (desempenho de C = 1,0).

	Fortran gcc 5.1.1	Julia 0.4.0	Python 3.4.3	R 3.2.2	Matlab R2015b	Octave 4.0.0	Mathematica 10.2.0	JavaScript V8 3.28.71.19	Go go1.5	LuaJIT gsl-shell 2.3.1	Java 1.8.0_45
fib	0.70	2.11	77.76	533.52	26.89	9324.35	118.53	3.36	1.86	1.71	1.21
parse_int	5.05	1.45	17.02	45.73	802.52	9501.44	15.02	6.06	1.20	5.77	3.35
quicksort	1.31	1.15	32.89	264.54	4.92	1866.01	43.23	2.70	1.29	2.03	2.60
mandel	0.81	0.79	15.32	53.16	7.58	451.81	5.13	0.66	1.11	0.67	1.35
pi_sum	1.00	1.00	21.99	9.56	1.00	299.31	1.69	1.01	1.00	1.00	1.00
rand_mat_stat	1.45	1.66	17.93	14.56	14.52	30.93	5.95	2.30	2.96	3.27	3.92
rand_mat_mul	3.48	1.02	1.14	1.57	1.12	1.12	1.30	15.07	1.42	1.16	2.36

Fonte: julialang.org.

O *benchmark* é um procedimento (técnica) utilizando um conjunto de programas e arquivos definidos para avaliar o desempenho do *hardware* e do *software* de um computador em uma dada configuração (SAWAYA, 1999). O termo "*benchmark*" é também empregado para designar os próprios programas (de *benchmarking*) desenvolvidos para executar os testes.

Diante do exposto, este trabalho foi desenvolvido com a proposta de apresentar a linguagem Julia como uma alternativa ao MATLAB®, a qual pode ser aplicada na pesquisa científica e no ensino de programação simbólica e numérica. Para isso foram comparadas algumas das características importantes das duas linguagens, realizando um *benchmark* no qual foram testados cinco algoritmos diferentes a fim de avaliar o tempo de execução e a exatidão dos resultados. Dessa forma, o artigo foi dividido nos seguintes tópicos: linguagens de programação avaliadas, comparativo entre as linguagens, códigos utilizados nos testes de desempenho, testes de *benchmark* e conclusão.

2 Linguagens de programação avaliadas

Nesta seção será apresentada uma breve descrição das linguagens de programação e suas vantagens e desvantagens.

2.1 Julia

Julia é uma linguagem de programação compilada (JIT – *Just In Time*) open source de alto nível, projetada com foco na computação científica e numérica de alto desempenho (BEZANSON *et al.*, 2015). Essa linguagem é relativamente jovem, posto que teve início no MIT em agosto de 2009 e, em fevereiro de 2012, tornou-se open source. É fruto do trabalho de quatro pesquisadores: Stefan Karpinski, Jeff Bezanson, Alan Edelman e Viral Shah (BEZANSON *et al,*. 2015).

A linguagem Julia foi pensada como uma linguagem para computação científica suficientemente rápida, tal como as linguagens C e Fortran, mas igualmente fácil de aprender como o MATLAB® e o Matemática®, com o objetivo de facilitar a modelagem computacional. É escrito em C, C++ e Scheme, e a biblioteca padrão é escrita utilizando a própria linguagem Julia. Possui forte inspiração em MATLAB®, Lisp, C, Fortran, Mathematica®, Python, Perl, R, Ruby, Lua, além de compartilhar muitas características de Dylan e Fortress.

Podem ser citadas como vantagens da plataforma MATLAB®:

- Fácil de utilizar para a escrita e o teste de programas;
- Diversos livros, cursos e muito material traduzido disponível na Internet;
- Grande número de toolboxes;
- Utilizado por uma vasta comunidade;
- Pode ser utilizado para construir aplicações do tipo GUI.

Como desvantagens da plataforma MATLAB® podem ser citadas:

- É uma linguagem interpretada, logo a execução é em muitos casos, mais lenta que as linguagens compiladas;
- Alto custo de licenças mesmo para uso pessoal;
- Códigos no formato ".m" não são plenamente compatíveis em outros ambientes de programação com suporte a este formato de arquivo.

3 Comparativo entre as linguagens

A linguagem de programação Julia, de acordo com os seus desenvolvedores, não foi criada para ser uma "cópia" do MATLAB®, porém, muitas características de sua arquitetura são similares, onde apresenta muitos comandos iguais sintaticamente e de forma de utilização. Pode-se entender isso como uma vantagem para atrair novos usuários adaptados ao MATLAB®, ao OCTAVE e ao SCILAB. Assim, nesta seção foram comparados aspectos importantes das duas linguagens, ilustradas nos Quadro 1 e 2.

Quadro 1 – Sintaxe dos comandos em MatLab®.

Variáveis e Constantes	varM = valor
Vetores e Matrizes	Vetor_M = [1 2 3], Matriz_M = [1 2 3; 4 5 6]

Laços de Repetição	if condição
	  instruções
	elseif condição
	  instruções
	else
	  instruções
	end
	
	for variável = vetor
	  instruções;
	end
	
	while condicao
	  instruções;
	end

Funções	function varR = nome(var) varR = instrução_var end
Entrada e Saída	load('dados.dat') save('arquivo.dat','dados_variavel','-ascii')

Quadro 2 – Sintaxe dos comandos em Julia.

Variáveis e Constantes	varJ = valor
Vetores e Matrizes	Vetor_J = [1 2 3], Matriz_J = [1 2 3; 4 5 6]

Laços de Repetição	if condição
	  instruções
	elseif condicao
	  instruções
	else
	  instruções
	end
	
	for variável = vetor
	  instruções
	end
	
	while condicao
	  instruções
	end

Funções	function nome(var) return instrução_var end
Entrada e Saída	open("arquivo.dat", "w") write("arquivo.dat",dados_variavel)

Conforme observado nos quadros (Quadro 1 e Quadro 2), a sintaxe dos comandos entre as duas linguagens apresenta muitas semelhanças. Diferença significativa somente na definição de funções.

3.1 Modelo de execução, paradigma principal e modelo de tipo de dados

Novamente, a linguagem Julia apresenta grande semelhança com MATLAB® com única diferença quanto ao modo de execução, como ilustrado no Quadro 3, sendo a linguagem Julia compilada e o MATLAB® interpretada.

Quadro 3 – Modelo de execução, paradigma principal e modelo de tipo de dados.

	JULIA	MATLAB®
Modelo de execução	Compilada Just in time	Interpretada
Paradigma principal	Múltiplos paradigmas	Múltiplos paradigmas
Modelo de tipo de dados	Dinâmico	Dinâmico

4 Códigos utilizados nos testes de desempenho

Para testes de desempenho, foram implementados em Julia e em MATLAB® os algoritmos de série de Fibonacci recursiva e a geração de um gráfico 2D dos polinômios de Berstein, utilizando as funções nativas de cálculo simbólico, algébrico e a equação diferencial numérica Runge-Kutta 45.

4.1 Série de Fibonacci recursiva

O algoritmo recursivo da série de Fibonacci é uma sequência onde cada termo subsequente, corresponde à soma dos dois anteriores. A série de Fibonacci recursiva é comumente utilizada em testes de desempenho, devido à simplicidade do código e o processamento de números inteiros trabalhados na forma de pilha, onde as implementações da série de Fibonacci em Julia e MATLAB® são ilustradas nos Quadros 4 e 5, respectivamente.

Quadro 4 – Código função série de Fibonacci em Julia.

```
function fiboR_JL(n)
   if n < 2
      return n
   else
fiboR_JL(n-1) + fiboR_JL(n-2)
   end
end
```

Quadro 5 – Código função série de Fibonacci em MATLAB®.

```
function F = fiboR_ML(n)
   if n < 2
        F = n;
   else
      F = fiboR_ML(n-1) + fiboR_ML(n-2);
   end
```

4.2 Operações de álgebra linear

A álgebra linear ocupa lugar de destaque nas diversas áreas da matemática (PESCADOR; POSSAMAI; POSSAMAI; 2013). Dessa forma, esse este teste visa explorar vários cálculos entre duas matrizes, sendo uma do tipo float e outra do tipo complexa, de tamanho $\boldsymbol{n \; x \; n}$, criadas de forma randômica em um laço for de 2 a $\boldsymbol{n}$. Os cálculos aplicados às duas matrizes são: traço de uma matriz quadrada, soma e multiplicação de elementos entre matrizes, inversão, soma de elementos de uma matriz, norma, autovalores, autovetores e a decomposição LU e QR. As implementações das operações matriciais em Julia e MATLAB® são ilustradas nos Quadros 6 e 7, respectivamente.

Quadro 6 – Código função operações de álgebra linear em Julia.

```
function algLin_JL(n)
   for i = 2:n
   A = complex(rand(i,i)); B = rand(i,i);
   A+B;
   trace(A);trace(B);
   A'; B';
   sum(A);sum(B);
   norm(A);norm(B);
   V,D = eig(A); V,D = eig(B);
A*B; A\B;
   La,Ua,Pa = lu(A); Lb,Ub,Pb = lu(B);
   Q,R = qr(A); Q,R = qr(B);
A.*B; B.*A;
   end
end
```

Quadro 7 – Código função operações de álgebra linear em MATLAB®.

```
function algLin_M(n)
for i = 2:n
    A = complex(rand(i,i));
   B = rand(i,i);
   A+B;
   trace(A);  trace(B);
   A';   B';
   sum(A);   sum(B);
   norm(A);  norm(B);
   [V,D] = eig(A);  [V,D] = eig(B);
A*B;A\B;
   [La,Ua,Pa] = lu(A);  [Lb,Ub,Pb] = lu(B);
   [Q,R] = qr(A);  [Q,R] = qr(B);
A.*B;  B.*A;
   end
end
```

4.3 Gráfico 2D dos polinômios de Berstein

Os polinômios de Bernstein apresentam a seguinte forma:

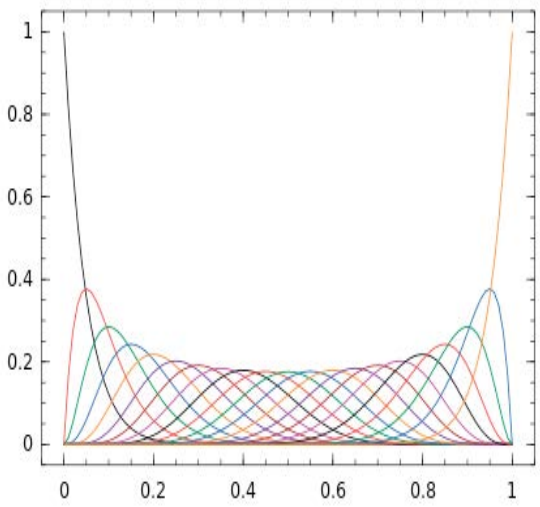
$$B_i^n = \binom{n}{i} x^i (1-x)^{n-i} \qquad (1)$$

O termo $\{B_i^n\}_{i=0}^n$ forma uma base para os polinômios de grau 1 até n. Se $P(x)$ é um polinômio de grau menor ou igual a n, então pode ser escrito como:

$$P(x) = \sum_{i=0}^{n} \beta_i \binom{n}{i} x^i (1-x)^{n-} \qquad (2)$$

O código de teste utiliza os polinômios de grau 1 até n para gerar várias linhas em uma única figura. O objetivo é verificar o desempenho do processamento de número flutuante para uma função polinomial, calculada elemento por elemento de um vetor, e a manipulação de dados é realizada para gerar um gráfico 2D a partir do resultado dos cálculos. O código dos polinômios de Berstein e como gerar tais gráficos são ilustrados nos Quadros 8 e 9 para o Matlab® e Julia, respectivamente.

Figura 1 – Gráfico da figura de Bernstein de grau 20.



Quadro 8 – Código dos polinômios de Bernstein e função plot em MATLAB®.

```
function B = bernstein_M(x,n)
B = zeros(length(x),n+1);
B(:,1) = 1;
   for k = 2:n+1
      B(:,k) = x.*B(:,k-1);
for j = (k-1:-1:2)
         B(:,j) = x.*B(:,j-1) + (1-x).*B(:,j);
      end
      B(:,1) = (1-x).*B(:,1);
   end
end

function plot_M(n)
  x = 0:1/100:1;  A = 0;
     for grau = 1:n
        plot(x, bernstein_M(x,grau));
    end
end
```

Quadro 9 – Código dos polinômios de Bernstein e função *plot* em Julia.

```
function bernstein_JL(x,n)
B = zeros(length(x),n+1)
B[:,1] = 1
   for k = 2:n+1
      B[:,k] = x.*B[:,k-1]
      for j = k-1:-1:2
         B[:,j] = x.*B[:,j-1] + (1-x).*B[:,j]
      end
      B[:,1] = (1-x).*B[:,1]
   end
   return B
end

function plot_JL(n)
x = 0:1/100:1; A = 0
for grau = 1:n
        A =plot(x,bernstein_JL(x,grau))
end
   display(A)
end
```

4.4 Cálculo simbólico

O cálculo simbólico é uma importante ferramenta para encontrar respostas analíticas de equações e demonstrações. O MATLAB® utiliza MuPad como motor de cálculo simbólico e a linguagem Julia utiliza a biblioteca Python SymPy. Para verificar o desempenho de processamento e exatidão no cálculo simbólico, foram utilizados os seguintes testes:

Derivada indefinida: Um laço for realiza cinco derivadas indefinidas para a função:

$$f(x) = sin(x+1)^2\left(\frac{csc(x+1)}{\sqrt{x-1}}\right) \quad (3)$$

Equação diferencial: Solução da equação diferencial com valores iniciais:

Teste algébrico: expansão e fatorização da expressão algébrica:

$$2y'' + 5y' + 20y = 0\,, y(0) = 1\, e\, y'(0) = 0 \quad (4)$$

Os Quadros 10 e 11 ilustram os códigos das funções de cálculo simbólico para Julia e o Matlab®, respectivamente.

Quadro 10 – Código das funções de cálculo simbólico em Julia.

```
using SymPy
@syms x
for i = 1:5
    diff(sin(x+1)^2*csc(x+1)/sqrt(x-1),x,i)
end
```

```
using SymPy
@syms x
y = SymFunction("y")
dsolve(y''(x)*2 + y'(x)*5+20*y(x), x, (y, 0, 1), (y', 0, 0))
```

```
using SymPy;
@syms x y
yf = ((x -1)*(y +1)/(x+y))^20
yex = expand(yf); yfc = factor(yex)
```

Quadro 11 – Código das funções de cálculo simbólico em MATLAB®.

```
syms x y
for i = 1:5
   diff(sin(x+1)^2*csc(x+1)/sqrt(x-1),x,i);
end
```

```
syms y(x)
Dy = diff(y);
D2y = diff(y,2);
dsolve(D2y*2 + Dy*5+20*y, y(0) == 1 , Dy(0) == 0);
```

```
syms x y
yf = ((x -1)*(y +1)/(x+y))^20;
yex = expand(yf); yfc = factor(yex);
```

4.5 Equação diferencial numérica usando Runge-Kutta 45

O método de Runge-Kutta é um dos métodos mais populares para a integração da equação diferencial de primeira ordem. Em problemas não-rígidos, é a melhor função a ser usada como primeira tentativa (GILAT; SUBRAMANIAM, 2008). Dessa forma, foi utilizada a função ODE45 própria do MATLAB® (ilustrada no Quadro 12) e ODE45 (pacote ODE) da linguagem JULIA (Quadro 13) para resolver uma equação diferencial por meio dos métodos de Runge-Kutta explícitos de quarta e quinta ordem.

Quadro 12 – Código das funções Runge-Kutta 45 em MATLAB®.

```
function [X1,Y1] = RK_M(n)
  f = @(x,y) (-5*x - y/5).^3 + 10;
  tspan = 0:0.001:n;
y0 = [0.0 1.0];
  [X1,Y1]= ode45(f,tspan,y0);
end
```

Quadro 13 – Código das funções Runge-Kutta 45 em Julia.

```
function Rk_JL(n)
  f(x,y)= (-5*x - y/5).^3 + 10
tspan = 0:0.001:n
 y0 = [0.0, 1.0]
  return ODE.ode45(f, y0,tspan);
end
```

5 Testes de *benchmark*

Neste trabalho, a técnica de *benchmark* foi utilizada para analisar o desempenho das linguagens de programação Julia e MATLAB® em relação ao tempo de execução e exatidão dos cálculos. O tempo de execução resultante corresponde à média aritmética simples de cinquenta repetições dos tempos de execução de cada teste, e a exatidão dos cálculos corresponde à comparação dos resultados numéricos/simbólicos obtidos. Os dados trabalhados são do tipo float, para os testes com números reais, e do tipo *complex*, para os testes com números complexos, limitados em seis casas decimais. Foi utilizado a linguagem Julia 0.4.5/Jupyter 4.1 e MATLAB® 2016 Versão *Trial*. O computador utilizado nos testes possui a configuração ilustrada no Quadro 14.

Quadro 14 – Configuração do computador de testes.

Sistema operacional Microsoft Windows 7 Professional
Processador QuadCore Intel Xeon W3520, 2800 MHz
Placa mãe Hewlett-Packard HP Z400 Workstation
Adaptador gráfico NVIDIA FX 380 (256 MB)
Memória RAM 6GB DDR-3 1066 MHz
Disco rígido ST31000528AS (1 TB, 7200 RPM, SATA-II)

O tempo de execução de uma função corresponde ao comando "@elapsed função" na linguagem Julia e tic e toc e no MATLAB®.

5.1 Série de Fibonacci recursiva

O MATLAB® e a linguagem Julia apresentaram o mesmo resultado numérico, no entanto, Julia obteve o menor tempo de execução em todos os testes. Para o termo 20 da série de Fibonacci, o MATLAB® apresentou o pior tempo de execução, sendo 42,740 vezes superior em relação ao tempo de execução da linguagem Julia. Esses testes estão ilustrados nas Tabelas 2 e 3.

Tabela 2 – Resultado do teste "série de Fibonacci recursiva" implementada em linguagem Julia.

Termo	Tempo decorrido (s)	Resultado
20	0,000073	6765
25	0,000811	75025
30	0,008753	832040
35	0,096720	9227465
40	1,046905	102334155

Tabela 3 – Resultado do teste "Série de Fibonacci Recursiva" no MATLAB®.

Termo	Tempo decorrido (s)	Resultado
20	0,003120	6765
25	0,024648	75025
30	0,173785	832040
35	1,894164	9227465
40	2,456066	102334155

5.2 Operações algébricas

O MATLAB® e a linguagem Julia apresentaram o mesmo resultado numérico (omitido em função do espaço). A linguagem Julia apresentou tempo de execução 5,650 vezes superior em relação ao tempo de execução do MATLAB® para n igual a 10. Enquanto que o MATLAB® obteve o pior tempo de execução de 2,103 vezes superior em relação ao tempo de execução da linguagem Julia para n igual a 50. Os resultados estão indicados na Tabela 4.

Tabela 4 – Resultado do teste "operações algébricas" em Matlab® e em linguagem Julia.

n	Tempo decorrido (s) – Matlab®	Tempo decorrido (s) – Julia
10	0,003432	0,019392
50	0,552244	0,262591
100	4,024514	2,626588
150	16,135495	9,821897
200	46,479250	25,103150

5.3 Gráfico 2D com figuras de Bernstein

O MATLAB® e a linguagem Julia apresentaram o mesmo conjunto de figuras, omitidas em função do espaço. De acordo com os resultados indicados na Tabela 5, a linguagem Julia obteve desempenho superior em todos os testes. O pior tempo de execução do MATLAB® ocorreu para o polinômio de grau 10, no qual foi 1,845 vezes superior em relação ao mesmo tempo de execução da linguagem Julia.

Tabela 5 – Resultado do teste "gráfico 2D com figuras de Bernstein" em Matlab® e em linguagem Julia.

Grau	Tempo decorrido (s) – Matlab®	Tempo decorrido (s) – Julia
10	0,090169	0,049132
20	0,158497	0,097846
30	0,310130	0,182600
40	0,521979	0,313305
50	0,730709	0,506201

5.4 Cálculo para matemática simbólica

Para a execução das equações (3)-(5), o MATLAB® e a linguagem Julia apresentaram o mesmo resultado simbólico, omitido em função do espaço. De acordo com os resultados indicados na Tabela 6, o MATLAB® obteve tempo de execução de 4,942 vezes superior em relação ao tempo de execução da linguagem Julia para o teste "Equação diferencial". A linguagem Julia apresentou nos testes "derivada indefinida" e "Expressão" tempo de execução de 2,682 e 38,223 vezes superior em relação ao tempo de execução do MATLAB®.

Tabela 6 – Resultado do teste "cálculo matemática simbólica" em Matlab® e em linguagem Julia.

Teste	Tempo decorrido (s) – Matlab®	Tempo decorrido (s) – Julia
Derivada indefinida	0,377210	1,011773
Equação diferencial	0,789365	0,159718
Expressão algébrica	0,302954	11,579924

5.5 Equação diferencial numérica usando Runge-Kutta 45

O MATLAB® obteve o menor tempo de execução em todos os testes de execução da equação diferencial numérica usando o método de Runge-Kutta 45, de acordo com os resultados indicados na Tabelas 7 e 8. Para n igual a 10, a linguagem Julia apresentou o pior tempo de execução, sendo 7,463 vezes superior em relação ao tempo de execução do MATLAB®. Os resultados numéricos apresentam diferença em função, em parte, das tolerâncias definidas pelos métodos de cada linguagem de programação. A linguagem Julia utiliza reltol = 1e-5 e abstol = 1e-8 enquanto que o MATLAB® utiliza reltol = 1e-3 e abstol = 1e-6.

Tabela 7 – Resultado do teste "equação diferencial numérica usando Runge-Kutta 45" em linguagem Julia.

n	Tempo decorrido (s)	Resultado
10	0,023284	-233,644744 / -233,644731
50	0,038995	-1233,644885 / -1233,644837
100	0,081779	-2483,645087 / -2483,644984
500	0,554714	-12483,644692 / -12483,644669
1000	1,208113	-24983,801762 / -24983,644668

Tabela 8 – Resultado do teste "equação diferencial numérica usando Runge-Kutta 45" no MATLAB®.

n	Tempo decorrido (s)	Resultado
10	0,003120	-233,656273 / -233,651020
50	0,010920	-1234,038994 /-1233,644669
100	0,019968	-2483,646188 / -2483,644668
500	0,120121	-12481,420316 / -12483,644668
1000	0,231817	-24983,540474 / -24983,540474

6 Conclusão

A linguagem de programação JULIA apresenta sintaxe muito próxima à do MATLAB®, além de compartilhar outras características importantes. Destaca-se pelo fato de ser livre e open source (licença MIT), não exigindo máquinas robustas e várias opções de IDE.

A linguagem de programação Julia apresentou menor tempo de execução para a maioria dos testes em uma relação de 15/23, contra 8/23 do MATLAB®, no qual o cálculo recursivo foi o principal destaque.

De acordo com o trabalho apresentado, a linguagem de programação Julia, devido sua simplicidade de sintaxe e resultados nos testes, apresenta potencial significativo como alternativa ao MATLAB® no campo da pesquisa e do ensino. Apesar de jovem e de ainda dispor de poucos pacotes disponíveis, a linguagem evolui a cada ano no ranking "TIOBE Index".

REFERÊNCIAS

BEZANSON, J. *et al.* **Julia Language Documentation**. Disponível em: <http://www.julialang.org>. Acesso em: 20 de Maio. 2016.

PESCADOR, J; POSSAMAI, J. P.; POSSAMAI, C. R. **Aplicação de Álgebra Linear na Engenharia**. In: XXXIX – Congresso Brasileiro de Educação em Engenharia (COBENGE 2011). Blumenau (SC): FURB. Anais. 2011.

GILAT, A.; SUBRAMANIAM, V. **Métodos numéricos para engenheiros e cientistas: uma introdução com aplicações usando o MATLAB**. Porto Alegre: Bookman, 2008.

SPERANDIO, D.; MENDES, J. T.; SILVA, L. H. M. **Cálculo numérico:** características matemáticas e computacionais dos métodos numéricos. São Paulo: Pearson, 2003.

SAWAYA, M. R. **Dicionário de informática & Internet**: inglês / português. São Paulo: Nobel, 1999.

AGRADECIMENTOS

Agradecemos o apoio da Universidade de Brasília (UnB) e do Colégio Técnico de Bom Jesus (CTBJ/UFPI).