УДК 004.43:[371.3+001.891]

Оконешникова Е. А.

НОВЫЙ ЯЗЫК ПРОГРАММИРОВАНИЯ В УЧЕБНОМ ПРОЦЕССЕ И НАУЧНЫХ ИССЛЕДОВАНИЯХ

Евгения Александровна Оконешникова

студентка факультета Математики и компьютерных наук, 4 курс

E-mail: owljaneok@yandex.ru

Кубанский Государственный университет

A NEW PROGRAMMING LANGUAGE IN THE EDUCATIONAL PROCESS AND RESEARCH

Evgeniya Aleksandrovna Okoneshnikova

Kuban State University

Аннотация. *Статья знакомит читателя с новым языком программирования Julia, приводятся некоторые его технические аспекты, сравнение с языками Python и Matlab. Демонстрируется пример применения данного языка в учебном процессе и научных исследованиях.*

Abstract. *The article introduces the new programming language Julia, provides some of its technical aspects, a comparison with the Python and Matlab languages. Demonstration of the use of this language in the educational process and scientific research.*

Ключевые слова: *научное программирование, язык программирования Julia, теория чисел, теорема Мертенса*

Keywords: *scientific programming, programming language Julia, number theory, Mertens theorem*

В процессе научных исследований часто возникает необходимость в проведении расчетов с высокой точностью, выполнении операций с числами больших порядков, обработке экспериментально полученных данных и проверке ста-

тистических гипотез. Вполне естественно для этих целей прибегнуть к вычислительным мощностям современных компьютеров. Поэтому перед исследователем возникает задача выбора языка программирования — универсального и достаточно простого для непрофессионалов. Этим требованиям отвечает язык программирования Julia (https://julialang.org/) - текущая версия Julia 1.3.1.

История создания языка Julia и его достоинства

Julia с февраля 2012 года разрабатывается в Массачусетском технологическом институте, который по многим рейтингам является несомненным мировым лидером в области IT-технологий. Авторы — Стефан Карпински, Джефф Безансон и Вирал Шах — стремились создать инструмент для проведения большого объема вычислений с максимально возможной производительностью. По этой причине в Julia предусмотрена поддержка облаков и параллельных вычислений. В августе 2018 г. вышла стабильная версия Julia 1.0 и язык стал стремительно развиваться.

Некоторые технические аспекты Julia

1) Julia имеет динамическую сильную неявную типизацию с детализированной системой типов, которая позволяет коду Julia работать так же быстро, как и C;

2) Julia — компилируемый язык с возможностью построчного выполнения в REPL;

3) Зарегистрировано более 2800 пакетов;

4) Заложена множественная диспетчеризация (мультиметод) функций. Это означает, что функция является объектом, который в зависимости от сообщенного списка аргументов выбирает надлежащий метод для вычисления значения функции;

5) Julia поддерживает параллельное программирование, а также работу с большими массивами данных;

6) Julia позволяет подключать модули и программы на C/C++, Fortran, Python и R. Другие языковые интерфейсы находятся в разработке.

Краткое описание синтаксиса. Сравнение с языками Python и Matlab

На синтаксис Julia большое влияние оказали языки Python и Matlab. Однако есть некоторые различия, которые делают Julia, на наш взгляд, более привлекательным для математиков. Рассмотрим несколько примеров (таблица 1).

Таблица 1 — Отличие синтекса Julia Python и Matlab

Julia	**Python**	**Matlab**
Задание вектора-строки 1x3		
A = [1 2 3]	A = numpy.array([1, 2, 3]).reshape(1, 3)	A = [1 2 3]
Задание вектора-столбца 3x1		
A = [1 2 3] ' или: A = [1; 2; 3]	A = numpy.array([1, 2, 3]).reshape(3, 1)	A = [1; 2; 3]
Обращение ко второму элементу вектора-строки и вектора-столбца		
A[1, 2] A[2, 1]	A[0, 1] A[1, 0]	A(0, 1) A(1, 0)
Задание матрицы 2x3		
M = [12 34 56; 78 91 23] или: M = [12 34 56 78 91 23]	M = numpy.array([[12, 32, 56], [78, 91, 23]])	M = [12 34 56; 78 91 23]
Обращение к элементу, стоящему на пересечении 2-й строки и 3-го столбца		
M[2,3]	M[1,2]	M[1,2]
Задание матрицы из нулей 2x3		
Вывести второй столбец матрицы		
Возведение матрицы в степень		
	numpy.linalg.matrix_power(A, 2)	

Как видно из примеров, в Julia индексация начинается не с нуля, а с единицы, что облегчает работу с векторами и матрицами. Циклы с пред- и постусловием, условный оператор if(else) реализуются аналогично синтаксису Matlab.

В Julia есть символьные типы BigInt и BigFloat для работы с большими числами. Для операций с переменными как с элементами группы, поля или кольца подключается пакет Nemo. Для построения графиков — пакет Plots. Обработка большого объема данных из файлов с разделителями осуществляется

с помощью пакета DelimitedFiles. Подключение этих и других пакетов продемонстрировано ниже, в примерах программ.

Применение в науке. Алгоритм Ферма. Теорема Мертенса

Чтобы в полной мере оценить возможности и преимущества языка программирования, необходимо начать использовать его в практических целях.

Популярный и уже достаточно хорошо изученный криптографический алгоритм RSA построен на вычислительной сложности задачи факторизации. Проблемой проверки чисел на простоту занимались известные математики Эйлер, Ферма, Гаусс, Лежандр и Мертенс. Ферма предложил свой метод разложения числа на пару множителей. Идея метода состоит в представлении числа в виде разности квадратов, а соответствующий алгоритм легко программируется. Реализация на языке Julia представлена на рисунках 1 и 2.

```
julia> n = 2124679;

julia> x = div(sqrt(n),1);

julia> if n == x^2
           println("$n - квадрат числа $x")
       end

julia> x +=1;

julia> while x < (n+1)/2
           global y = sqrt(x^2 - n);
           y == div(y,1) ? break : global x+=1;
       end

julia> x == (n+1)/2 ? println("$n - простое число") : println("p = $(x+y), q = $(x-y)")
2124679 - простое число
```

Рисунок 1 — Алгоритм Ферма проверки на простоту

```
julia> n = 7642656;

julia> x = div(sqrt(n),1);

julia> if n == x^2
           println("$n - квадрат числа $x")
       end

julia> x +=1;

julia> while x < (n+1)/2
           global y = sqrt(x^2 - n);
           y == div(y,1) ? break : global x+=1;
       end

julia> x == (n+1)/2 ? println("$n - простое число") : println("p = $(x+y), q = $(x-y)")
p = 2856.0, q = 2676.0
```

Рисунок 2 — Алгоритм Ферма проверки на простоту

Теория чисел — это область математики, в которой порой нет способа проверки справедливости гипотезы, кроме эмпирического. Выполнение математических расчетов вручную отнимало драгоценные время и силы гениев. В наше время, благодаря вычислительным мощностям и производительности современных компьютеров и суперкомпьютеров, стали возможны операции с числами в сотни и тысячи знаков. Так получила развитие экспериментальная теория чисел.

Рассмотрим известную теорему Мертенса [1] о сумме значений функции Эйлера:

$$\sum_{i=0}^{n} \varphi(i) = \frac{3}{\pi^2} n^2 + O(n \ln(n))$$

Для приложений необходима более точная оценка значения этой суммы для конкретного числа. В статье [2] показана справедливость следующей теоремы: *имеет место равенство*

$$\sum_{i=0}^{n} \varphi(i) = \frac{3}{\pi^2} n(n+1) + n \cdot s(n),$$

где для всех $n < 1{,}471 \cdot 10^{12}$ выполняется неравенство |s(n)| < 0,45.

Были проведены вычисления на Julia до 1.66 трлн., и выяснилось, что отклонение от поправленного среднего значения функции Эйлера походит на случайную величину, имеющую среднее значение 0.

В процессе вычислений в файл были выведены значения s(n) по абсолютной величине превосходящие 0,43 и числа n, на которых они достигались. Полученные данные мы решили проанализировать, чтобы выяснить, как распределены эти экстремальные значения.

Для начала мы нормировали массив чисел на 10^{10}. Затем были вычислены мат. ожидание и дисперсия расстояний между числами, для которых $|s(n)| > 0{,}43$. На рисунке 3 представлена соответствующая программа, которая считывает данные из файла с разделителем и проводит вычисления с высокой точностью. Результаты приведены в закомментированных строках.

```
using DelimitedFiles
data = convert.(BigFloat, DelimitedFiles.readdlm("data.csv", ','))
n = append!([0.],data[:,1]./10^10) #Экстремальные значения n
distans = Array{BigFloat,1}();

for i in 2:length(n)              #Запись вектора расстояний между n
    r = BigFloat(n[i])-BigFloat(n[i-1]);
    push!(distans, r);
end
distans

M=BigFloat(0);
for i in 1:length(distans)
    global M+= distans[i];
end

M = BigFloat(M/length(distans))
# M = 1.790521636155319279072152331788489159117353723404255319148936170212765957446806

D=0;
for i in 1:length(distans)
     global D+= distans[i]^2;
end

D = BigFloat(D/length(distans)) - BigFloat(M^2)
# D = 3.521250404141371192130762946612654791346161376735493423155573612771868357945 45
```

Рисунок 3 — Программа на julia, вычисляющая мат.ожидание и дисперсию расстояний между числами, на которых достигаются экстремальные отклонения

Затем были найдены значения функции распределения (рисунок 4) и построен график плотности с шагом 0,5 млрд. (рисунок 5).

```
R = map(x->x/10^10,distans)   #Вектор нормированных расстояний
B = sort(R)

F = Vector{Float64}()  #значения функции распределения расстояний
i = 1;
for c in 1:212
    while B[i]<0.05*c
        global i += 1;
    end
    push!(F,(i-1)/length(B));
end
F = [0
     F
     1]

dF = Array{BigFloat,1}() #Вектор приращений функции распределения
for i in 2:length(F)
    df = BigFloat(F[i])-BigFloat(F[i-1]);
    push!(dF, df);
end

using Plots
x = range(0.1, length = length(dF), step=0.05)
plot(x,dF./0.05; legend = false, xlabel = "x", ylabel = "f(x)")
```

Рисунок 4 — Код программы на julia для построения графика плотности

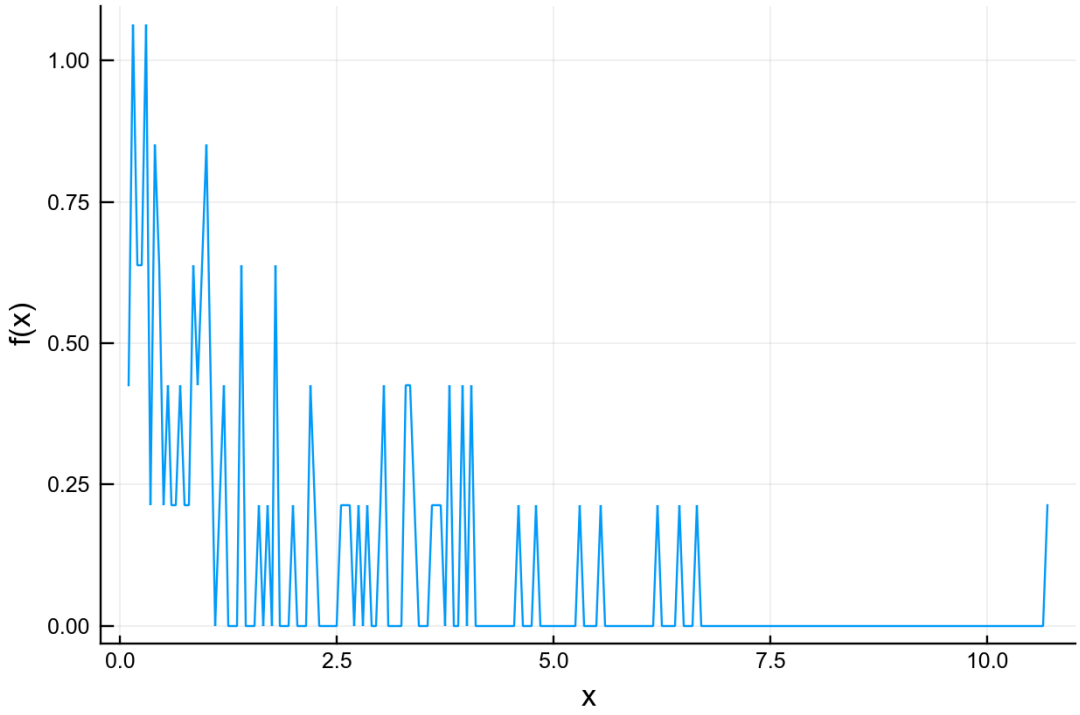

Рисунок 5 — График плотности распределения экстремальных отклонений

На сегодняшний день к сообществу пользователей Julia примыкают люди со всех уголков планеты. По результатам опроса, которые были представлены на конференции JuliaCon в июне 2019 года, 93 % опрошенных дают положительную оценку языку Julia, из них 73 % причисляют его к своим фаворитам среди языков. В числе плюсов технических характеристик респонденты называют скорость и производительность, простоту использования, отрытый код и мультиметод функций. К минусам относят незрелость некоторых пакетов и длительность генерации первого графика. Ознакомиться с полным отчетом можно по ссылке https://julialang.org/images/2019-julia-user-developer-survey.pdf.

Список литературы

1. Чандрасекхаран, К. Введение в аналитическую теорию чисел. / К. Чандрасекхаран. Москва : Мир, 1974. – 178 с.

2. Рожков А. В. Экспериментальная теория чисел — вычисления в программной среде Julia — I / А. В. Рожков, Н. В. Потапова, Е. А. Оконешникова // Информационныен технологии в математике и математическом образовании : материалы научно-методической конференции / КГПУ им. В. П. Астафьева, Красноярск, 2019. – С. 10–15.