Вариативное задание 2.2
Практика
Подготовка аналитической подборки Julia

Julia – высокоуровневый язык, который разработан для научного программирования. Язык поддерживает широкий функционал для математических вычислений и работы с большими массивами данных. Это язык с динамической компиляцией, программы на Julia компилируются в быстрый нативный код для таких платформ, как Windows, macOS и Linux. Приложения на языке Julia работают так же быстро, как приложения, которые написаны на быстрых низкоуровневых языках, таких как C или C++. Синтаксически Julia похож на Python и MATLAB, но эти языки – интерпретируемые, поэтому приложения на Julia работают быстрее.

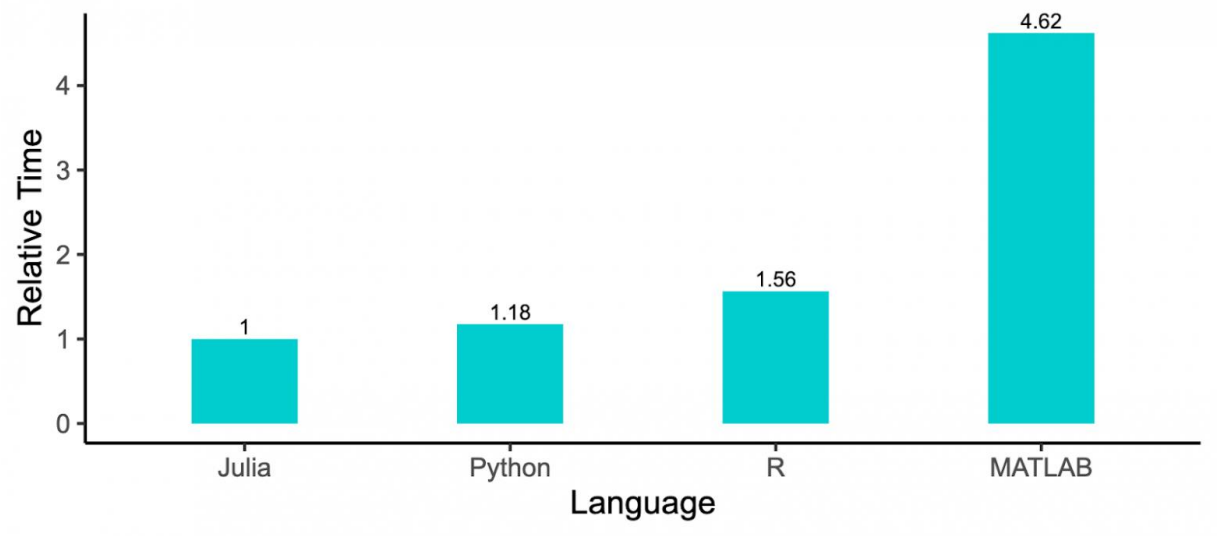
Работа над кодом Julia ведется в Juno – интегрированной среде разработки, которая является надстройкой над IDE Atom. Поскольку Juno рассчитана на работу с Julia, в интерфейсе рабочей среды технически реализованы возможности языка. Благодаря JIT-компиляции, разработчик может смотреть на вычисления определенных методов на ходу, а также следить за их компиляцией в машинный код. В Juno есть встроенная панель графиков, которая позволяет визуализировать функции Julia и даже анимировать графики с рендерингом в GIF. Поскольку Julia может работать с большими массивами данных, Juno поддерживает облачные вычисления и удаленный запуск на высокопроизводительных компьютерах.

Julia – это скорость

Одно из главных преимуществ Julia – ее скорость. Разработанная так, чтобы быть такой же быстрой, как языки, C и Fortran, Julia обеспечивает высокопроизводительные вычислительные возможности, которые так важны в современном мире интенсивных данных.

Высокая скорость работы Julia в первую очередь объясняется ее компилятором Just-In-Time (JIT). Это позволяет Julia компилировать эффективный нативный код, что делает его подходящим выбором для выполнения сложных алгоритмов на реальном оборудовании.

В контексте машинного обучения и глубокого обучения скорость Julia становится особенно выгодной. Она позволяет быстро обрабатывать большие массивы данных и эффективно выполнять тяжелые в вычислительном плане задачи, ускоряя тем самым темпы разработки ИИ.



Удобство Julia

Синтаксис Julia дружелюбен к пользователям Python и MATLAB, то есть на нем удобно программировать тем, кто уже знаком с этими языками. Читайте – если вы знаете Python/MATLAB, то вы практически знаете Julia, нужно только погрузиться. Также, если сравнивать синтаксис напрямую с MATLAB, то значительные различия можно пересчитать по пальцам.

Кроме того, высокоуровневый синтаксис позволяет легко выражать сложные алгоритмы, что делает язык более доступным и придает ему значительные выразительные возможности.

Синтаксис Julia интуитивно понятен и прост в изучении. Переменные можно присваивать без объявления их типа, язык поддерживает все распространенные алгоритмические структуры (циклы, условия), все распространенные структуры данных (многомерные матрицы, словари), а для сложных типов данных полно бесплатных библиотек.

Роль Julia в IT-трансформации науки

Высокоуровневый синтаксис Julia и эффективный JIT-компилятор позволяют Julia играть важную роль в активно развивающихся областях науки, например в области квантовых вычислений.

В одной из недавних статей было предложено использовать Julia в качестве инструмента для разработки алгоритмов для квантовых компьютеров. В статье был представлен Yao.jl, расширяемый и эффективный фреймворк для разработки квантовых алгоритмов, подчеркивающий потенциал Julia в области квантовых вычислений.

Yao.jl позволяет квантовым программистам разрабатывать и тестировать квантовые алгоритмы с помощью таких функций, как поддержка GPU и механизм автоматического дифференцирования. Пакет обещает самую

современную производительность, что еще больше подчеркивает потенциал Julia в области квантовых вычислений.

Возможности Julia:

- использование LLVM для компиляции в машинный код;
- большое количество математических функций;
- множественная диспетчеризация;
- реализация параллельных вычислений;
- поддержка метапрограммирования;
- встроенный менеджер пакетов;
- взаимодействие с языками C, Python, R, а также поддержка их пакетов;
- поддержка интроспекции кода.

Где применяют Julia

- математические вычисления;
- анализ больших массивов данных;
- машинное обучение;
- веб-разработка благодаря таким фреймворкам, как Genie.

Преимущества вычислений на Julia

- высокоуровневый, простой синтаксис;
- Juno – IDE, которая хорошо раскрывает возможности языка;
- хорошая производительность, которая близка к низкоуровневым языкам;
- регулярные обновления языка;
- есть вспомогательная информация, переведенная на русский.

Недостатки вычислений на Julia

- малое количество специалистов;
- баги, которые могут возникать из-за проблем с совместимостью разных версий языка и пакетов-расширений, от чего в вычислениях возникают ошибки;
- приложения на Julia занимают большой объем памяти;
- среда выполнения кода требует большой мощности процессора и оперативной памяти.

Примеры готовых работ на Julia

Посмотрим, что умеет этот калькулятор… Поддержка Юникода — можно использовать кириллицу, иероглифы и назвать pi греческой буквой. А еще

можно явно не указывать умножение между числом и переменной (именно в таком порядке и без пробела):

```
x = 5+8
2x - 3x + 2x^2
Out: 325
```

Все нужные знаки тоже на месте: +=, *=, >>= и т.д. (Знак ">>" (битовый сдвиг вправо). Знаки сравнения: >, >=, <, <=, ==, !=. Неравенства можно объединять в цепочки:

```
y = 5
y += 2
4 <= y < 8
Out: true
```

Пример реализации чисел Фибоначи:

```
function fib(n)
      if n == 0 return 0 end
      if n == 1 return 1 end
      return fib(n-1) + fib(n-2)
end
```

Пример вычисления факториала числа:

```
function factorial(n)
    if n == 0
        return 1
    else
        return n * factorial(n - 1)
    end
end

println(factorial(5))  # 120
```

Работать с матрицами:

```
A = [1 2 3; 6 5 4; 7 8 9]
Out: 3x3 Array{Int64,2}:
 1  2  3
 6  5  4
 7  8  9
A[2,1]
Out: 6
A[end]
Out: 9
size(A)
Out: (3, 3)
```

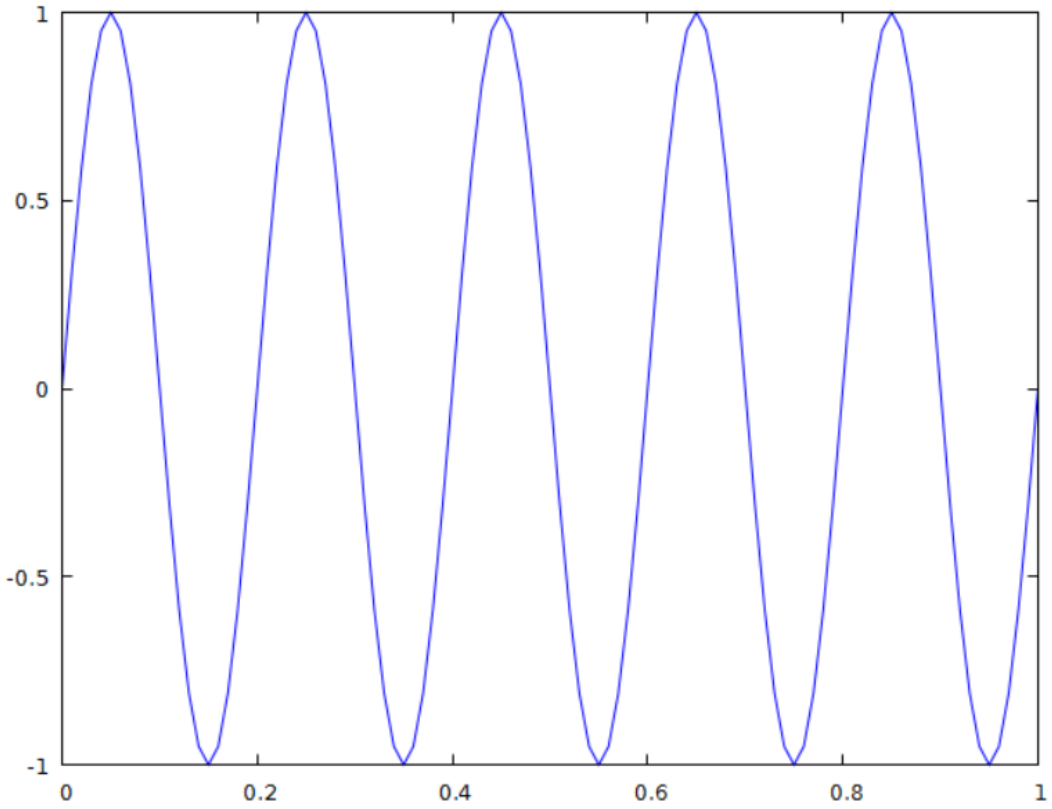
И т.д.

```
# неизменяемая структура данных
struct ImmutableVector
  data::Vector{Int}
end

# функция, создающая новый ImmutableVector с добавленным элементом
function add_element(ivec::ImmutableVector, element::Int)
  new_data = copy(ivec.data)
  push!(new_data, element)
  return ImmutableVector(new_data)
end

ivec = ImmutableVector([1, 2, 3])
ivec_new = add_element(ivec, 4)
```

```
using Gaston
t = 0:0.01:1
plot(t, sin.(2π*5*t))
```



Источники:

1. Шесть причин выучить Julia в 2024 году // Habr URL: https://habr.com/ru/companies/etmc_exponenta/articles/831670/ (дата обращения: 23.09.2024).
2. Кратко про FP в Julia // Habr URL: https://habr.com/ru/companies/otus/articles/794819/ (дата обращения: 23.09.2024).
3. Julia. Знакомство // Habr URL: https://habr.com/ru/articles/423811/ (дата обращения: 23.09.2024).
4. Что такое Julia // Workspace URL: https://workspace.ru/tools/language/julia/ (дата обращения: 23.09.2024).
5. О языке программирования Julia // Gitbooks URL: https://closescreen.gitbooks.io/julia-lang-ru/content/ (дата обращения: 23.09.2024).