DOI 10.37882/2223-2966.2023.09.11

ОБЗОР БИБЛИОТЕКИ DIFFERENTIALEQUATIONS.JL ДЛЯ РЕШЕНИЯ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ЧИСЛЕННЫМИ МЕТОДАМИ

OVERVIEW OF THE DIFFERENTIALEQUATIONS.JL LIBRARY FOR SOLVING DIFFERENTIAL EQUATIONS BY NUMERICAL METHODS

P. Komarov

Summary. The article provides an overview of one of the most powerful tools created for solving differential equations by numerical methods — the DifferentialEquations.jl library, created on the basis of the Julia programming language. Results: a) the essence of the Julia programming language is indicated and its key advantages are highlighted; b) the main functional capabilities of the DifferentialEquations.jl library are described; c) examples of solving differential equations by means of the DifferentialEquations.jl library are considered; d) the comparative analysis of the software allowing to solve differential equations is carried out; e) it is revealed that the DifferentialEquations.jl library reflects the advantages of numerous other products, but its documentation is quite voluminous.

Keywords: Julia, programming, differential equations, numerical methods, Differential Equations.jl library.

Комаров Петр Олегович
Аспирант, Российский университет дружбы народов (РУДН)
pokomarov98@gmail.com

Аннотация. В статье проводится обзор одного из наиболее мощных инструментов, созданных для решения дифференциальных уравнений численными методами — библиотеки DifferentialEquations.jl, созданной на базе языка программирования Julia. Результаты: а) указана сущность языка программирования Julia и выделены его ключевые преимущества; б) описаны основные функциональные возможности библиотеки DifferentialEquations.jl; в) рассмотрены примеры решения дифференциальных уравнений с помощью библиотеки DifferentialEquations.jl; г) проведён сравнительный анализ программного обеспечения, позволяющего решать дифференциальные уравнения; д) установлено, что в библиотеке DifferentialEquations.jl отражены сильные стороны иных многочисленных продуктов, однако её документация является довольно объёмной.

Ключевые слова: Julia, программирование, дифференциальные уравнения, численные методы, библиотека DifferentialEquations.jl.

Введение

Решение дифференциальных уравнений — актуальная повестка для математиков, физиков, астрономов, инженеров и иных специалистов, занятых расширением края передовых наук. На заре третьего десятилетия XXI в., когда искусственный интеллект, машинное обучение, нейронные сети и иные цифровые технологии становятся всё более востребованными, актуальность не теряет и необходимость решения дифференциальных уравнений, поскольку в современной науке многие непростые физические процессы описываются именно с их помощью [Ильичев, Юрик и Медов, 2022].

В последнее время популярность набирает относительно молодой язык программирования Julia, представляющий собой универсальный и уже достаточно зрелый [Кулябов и Королькова, С. 49], гибкий динамический язык программирования и научных вычислений [Зырянов, Емельянов и Лещенко, 2020], который может быть использован во многих интересных областях (проекты в области науки о данных, корпоративные приложения, работа с пространствами имен, модулями и пакетами и мн. др. [Kwong, 2020]). В частности, одной из особенностей Julia является функциональная возможность решения дифференциальных уравнений численными методами, заданная библиотекой DifferentialEquations.jl.

1. Сущность языка программирования Julia и его преимущества

Ни один из общепринятых языков (C/C++, Fortran, MATLAB или Python) не может сбалансировать производительность и эффективность. Альтернативный подход состоит в объединении статических и динамических языков: для того, чтобы преодолеть эту «проблему двух языков», в Массачусетском технологическом институте в 2009 г. была начата работа по разработке языка Julia с открытым исходным кодом. Первоначальное намерение языка Julia состояло в том, чтобы объединить производительность и эффективность [Bezanson et al, 2012].

Язык программирования Julia был выпущен в 2012 г. четырьмя экспертами: А. Эдельманом, С. Карпински, Дж. Безансоном и В. Шахом. На сегодняшний день актуальной версией данного языка является Julia 1.9. Это свободный язык программирования с открытым исходным кодом, который ориентирован в основном на вычисле-

ние научных, в том числе математических вычислений. Роль языка Julia заключается в том, что он позволяет устранить компромисс в производительности и предоставить единую среду, достаточно производительную для создания прототипов и достаточно эффективную для развертывания приложений, требовательных к производительности [Julia 1.9…, 2023].

Julia — это язык сценариев, который используется вместо таких языков, как R, Python, MATLAB, но предлагает производительность, которую можно ассоциировать с низкоуровневыми скомпилированными языками. Это позволяет пользователям запускать прототипы в Julia, а также решать свои крупномасштабные модели на одном языке, вместо того чтобы прибегать к решениям на двух языках, когда требуется производительность.

Julia достигает этой цели за счет широкого использования множественной диспетчеризации и метапрограммирования для разработки языка, который одновременно прост для понимания компилятором и прост в использовании для программиста [Bezanson et al, 2017].

Несмотря на то, что сегодня количество языков программирования является достаточно большим, и каждая компания или IT-специалист вправе самостоятельно выбирать тот язык, который лучше подходит для достижения целей деятельности, каждый из них обладает рядом уникальных преимуществ. Так, Б. Лаувенс и А.Б. Дауни выделяют следующие ключевые характеристики Julia:

— язык Julia разработан как высокопроизводительный язык программирования;
— язык Julia использует множественную диспетчеризацию, позволяющую программисту выбирать из различных шаблонов программирования, адаптированных к приложению. [Королькова, Геворкян и др., 2022, 2022].
— язык Julia — это динамически типизированный язык, который можно легко использовать в интерактивном режиме;
— у языка Julia приятный высокоуровневый синтаксис, который легко выучить;
— язык Julia — это необязательно типизированный язык программирования, чьи (определяемые пользователем) типы данных делают код более понятным и надежным;
— у языка Julia есть расширенная стандартная библиотека и доступно множество сторонних пакетов [Lauwens and Downey, 2019, P, 10].

Отечественные учёные также выделяют следующие преимущества использования Julia: удобную документацию; наличие макросов и иных возможностей метапрограммирования; поддержку юникода; наличие встроенного пакетного менеджера; открытость проекта, или open source [Зырянов, Емельянов и Лещенко, 2020, С. 55].

В целом следует отметить, что Julia объединяет преимущества других языков и фокусируется на научных вычислениях и анализе данных. Julia синтаксически похож на динамические языки, такие как MATLAB и Python, что делает его относительно простым в изучении. С точки зрения эффективности, Julia соответствует производительности C/C++ и FORTRAN и намного быстрее, чем MATLAB и Python.

2. Возможности библиотеки DifferentialEquations.jl

Дифференциальные уравнения являются фундаментальными компонентами многих научных моделей; они используются для описания крупномасштабных физических явлений, таких как планетарные системы и климат Земли, вплоть до биологических явлений меньшего масштаба, таких как биохимические реакции и процессы развития. Вследствие повсеместного распространения этих уравнений были разработаны стандартные наборы решателей, такие, как, например, набор Shampine ODE для MATLAB, коды Хайера на Фортране и решатели Sundials CVODE. Цель библиотеки DifferentialEquations.jl состояла в том, чтобы построить основу, созданную предыдущими библиотеками дифференциальных уравнений, и модернизировать их с помощью программного языка Julia [Xiao et al, 2022].

Решение дифференциальных уравнений с помощью Julia лежит в плоскости научного машинного обучения (SciML — Scientific Computing + Machine Learning). Организация SciML — это набор инструментов, предназначенных для решения уравнений и систем моделирования, разработанных на языке программирования Julia с возможностью привязки к иным языкам, таким как R и Python. Также SciML — это единая экосистема, состоящая из хорошо поддерживаемых инструментов. Она имеет согласованный принцип разработки, унифицированные API-интерфейсы для больших наборов решателей уравнений, всеобъемлющий анализ дифференцируемости и чувствительности, а также многие из самых высокопроизводительных и параллельных реализаций [Julia 1.9…, 2023]. Так, например, в отличие от большинства пакетов дифференциальных уравнений, которые требуют, чтобы пользователь понимал некоторые детали реализации библиотеки, экосистема DifferentialEquations.jl реализует различные предметно-ориентированные языки с помощью макросов, чтобы предоставить более естественные возможности для определения математических конструкций [Rackauckas and Nie, 2017].

DifferentialEquations.jl — это набор для численного решения дифференциальных уравнений, написанный на языке Julia и доступный для использования в Julia, Python и R. Целью этой библиотеки (пакета) является предоставление эффективных реализаций Julia для решателей различных дифференциальных уравнений.

DifferentialEquations.jl включает в себя современный набор решателей дифференциальных уравнений, предлагающих унифицированный пользовательский интерфейс для решения и анализа дифференциальных уравнений без ущерба для функций или производительности [Lindner, 2021]. Уравнения в области этого пакета включают:

- дискретные уравнения (функциональные карты, дискретное стохастическое моделирование);
- обыкновенные дифференциальные уравнения (ОДУ);
- разделенные и разделенные ОДУ (симплектические интеграторы, методы IMEX);
- стохастические обыкновенные дифференциальные уравнения (СОДУ или СДУ);
- стохастические дифференциально-алгебраические уравнения (СДАУ);
- случайные дифференциальные уравнения (RODE или RDE);
- дифференциальные алгебраические уравнения (ДАУ);
- дифференциальные уравнения с запаздыванием (DDE);
- нейтральные, запаздывающие и алгебраические дифференциальные уравнения с запаздыванием (NDDE, RDDE и DDAE);
- стохастические дифференциальные уравнения с запаздыванием (SDDE);
- экспериментальную поддержку стохастических нейтральных, запаздывающих и алгебраических дифференциальных уравнений с запаздыванием (SNDDE, SRDDE и SDDAE);
- смешанные дискретные и непрерывные уравнения (Hybrid Equations, Jump Diffusions);
- стохастические дифференциальные уравнения в частных производных ((S)PDE).

Кроме того, DifferentialEquations.jl имеет встроенные функции анализа, в том числе:

- прямой и сопряженный анализ чувствительности (автоматическое дифференцирование) для быстрых вычислений градиента;
- оценку параметров и байесовский анализ;
- нейронные дифференциальные уравнения с DiffEqFlux.jl для эффективного научного машинного обучения (научного машинного обучения) и научного искусственного интеллекта;
- автоматическое распределенное, многопоточное и графическое параллельное моделирование ансамбля;
- глобальный анализ чувствительности;
- количественную оценка неопределенности.

Благодаря широкому использованию множественной диспетчеризации, метапрограммирования, рецептов построения графиков, интерфейсов внешних функций (FFI) и перегрузки вызовов [Rackauckas and Nie, 2017], DifferentialEquations.jl предлагает унифицированный пользовательский интерфейс для решения и анализа различных форм дифференциальных уравнений без ущерба для функций или производительности.

3. Примеры решения дифференциальных уравнений с помощью DifferentialEquations.jl

Использование библиотеки DifferentialEquations.jl предполагает выполнение трёх шагов: определение проблемы; решение проблемы и построение решения [Julia 1.9…, 2023]. Самым простым примером использования численного метода посредством для построения решения является поиск простых чисел до 3 млн, представленный в работе А. В. Рожкова и А. В. Барсуковой [Рожков и Барсукова, 2022] (листинг 1).

Листинг 1: код программы поиска простых чисел до 3 млн

```
julia> using Nemo
Welcome to Nemo version
0.29.1
Nemo comes with absolutely no
warranty whatsoever
julia> function Ros(m,n)
    N = 1
        for i= m:n
            if
isprobable_prime(ZZ(2*i+1))
        N+=1
    end
    end
        print(N)
    end
Ros (generic function with
1 method)
julia> @time Ros(1,15*10^5)
216816 0.521483 seconds
1.51 м allocations:
23.168 мів
```

Так, посредством подключения алгебраического пакета Nemo и использования макроса @time был получен результат, согласно которому время вычислений составило всего лишь приблизительно полсекунды.

В качестве наиболее типичного примера решения дифференциальных уравнений с помощью DifferentialEquations.jl можно назвать знаменитую систему уравнений Лоренца (формулы 1–3):

$$\frac{dx}{dt} = \sigma\left(y - x\right) \quad (1)$$

$$\frac{dy}{dt} = x(\rho - z) - y \quad (2)$$

$$\frac{dz}{dt} = xy - \beta z \quad (3)$$

Для построение решения в Julia пользователь должен переписать эту функцию таком формате, который будет «понятен» компьютеру, определив u=[x;y;z] в качестве вектора и записав уравнение в терминах этого вектора. Формат ODE.jl, аналогичный другим языкам сценариев, таким как SciPy или MATLAB, выглядит следующим образом (листинг 2).

Хотя этот формат принимается в библиотеке DifferentialEquations.jl, предоставляются дополнительные макросы удобства использования, которые будут автоматически преобразовывать пользовательский ввод из более математического формата.

Так, для обыкновенного дифференциального уравнения предоставляется @ode_def, который позволяет пользователю определить то же самый обыкновенное дифференциальное уравнение следующим образом (листинг 3).

Листинг 2: Описание формата ODE.jl

```
f = (t,u,du) -> begin
du[1] = 10*(u[2]-u[1])
du[2] = u[1]*(28-u[3]) — u[2]
du[3] = u[1]*u[2] — 8/3*u[3]
end
```

Листинг 3: Описание макроса @ode_def

```
f = @ode_def Lorenz begin
dx = σ*(yx)
dy = x*(ρ-z) — y
dz = x*y — β*z
end σ=>10. p=>28. β=(8/3)
```

Поскольку Julia позволяет использовать юникод в коде, этот формат соответствует стилю, который можно было бы ожидать в публикации TeX. Макрос принимает это определение, находит значения для левой части формы «d__» и использует словарь для поиска или замены этих значений для записи функции в формате других библиотек языка сценариев. Таким образом, с помощью библиотеки DifferentialEquations.jl может быть выполнен перевод в векторную систему, что позволяет пользователям иметь более читаемые сценарии без ущерба для производительности.

Ещё одним примером является представленное на официальном сайте Julia моделирование внешней Солнечной системы на основе данных, представленных в книге Э. Хайрера, К. Любича и Г. Ваннера [Hairer, Lubich and Wanner, 1999].

В качестве выбранных единиц анализа выступили массы относительно солнца (солнце имеет массу 1), т.е. с учётом внутренних планет m_0 = 1.00000597682. Расстояния были выражены в астрономических единицах, а время — в земных сутках, поэтому гравитационная постоянная (G) равнялась $2{,}95912208286 \times 10^{-4}$. Представим исходные данные в таблице (табл. 1). Применяя

Таблица 1.

Исходные данные для моделирования внешней Солнечной системы с помощью DifferentialEquations.jl

Планета	Масса	Исходное положение	Начальная скорость
Юпитер	$m_1 = 0.000954786104043$	-3.5023653 -3.8169847 -1.5507963	0.00565429 -0.00412490 -0.00190589
Сатурн	$m_2 = 0.000285583733151$	9.0755314 -3.0458353 -1.6483708	0.00168318 0.00483525 0.00192462
Уран	$m_3 = 0.0000437273164546$	8.3101420 -16.2901086 -7.2521278	0.00354178 0.00137102 0.00055029
Нептун	$m_4 = 0.0000517759138449$	11.4707666 -25.7294829 -10.8169456	0.00288930 0.00114527 0.00039677
Плутон	$m_5 = 1/(1.3108)$	-15.5387357 -25.2225594 -3.1902382	0.00276725 -0.00170702 -0.00136504

Источник: [Simulating the Outer Solar System…, 2023]

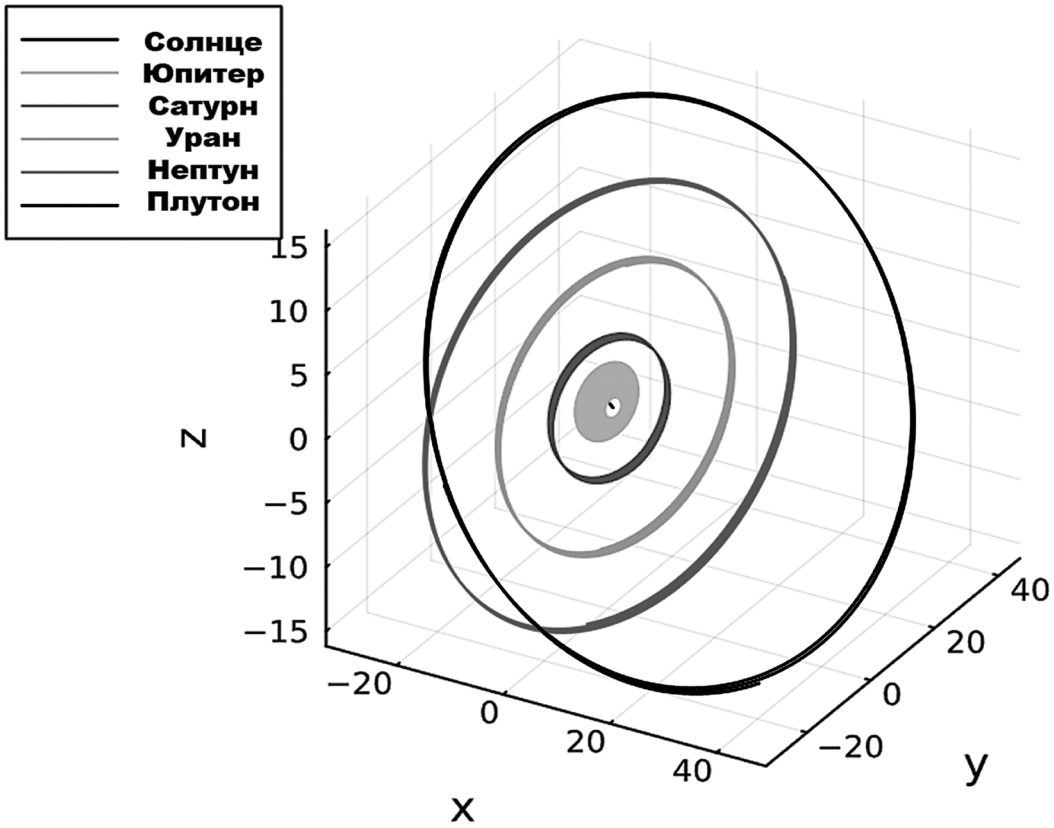

Рис. 1. Моделирование внешней Солнечной системы [Simulating the Outer Solar System…, 2023]

расчёты в области Гамильтоновой системы, в частности, гравитационную задачу N тел (N-body problem), можно на языке Julia построить проблему обычного дифференциального уравнения второго порядка и, запустив код, вывести решение, в том числе через стандартный plot показать движение пяти внешних планет относительно Солнца (рис. 1).

Таким образом, эти и другие примеры решения дифференциальных уравнений с помощью библиотеки (пакета) DifferentialEquations.jl наглядно демонстрируют возможности Julia.

4. Сравнение DifferentialEquations.jl с другими продуктами в области решения дифференциальных уравнений

Существующие пакеты программного обеспечения для различных научных задач написаны в основном на C/C++, Fortran, MATLAB или Python. Однако статические языки (C/C++ и Fortran) работают быстро, однако их относительно сложно использовать, и требуется длительное время для обучения, что не способствует их использованию непрофессиональными разработчиками. В тоже время динамические языки высокого уровня (Python и MATLAB) имеют больше преимуществ с точки зрения низких затрат на обучение, сильных визуализаций и взаимодействий, что привело к их преобладанию в современных научных вычислениях, однако они обычно оказываются неэффективными при решении ресурсоемких задач и при этом чрезмерно дорогостоящими.

Именно поэтому Julia обладает преимуществами обоих типов языков. Синтаксис этого языка достаточно прост, легок и выразителен, как и в языках высокого уровня. Кроме того, язык Julia упрощает утомительный синтаксис. Так, например, используя программу имитации полета, Р. Селлс сравнил количество кода, которое используется в Julia, с кодами других языков, и результаты показали, что количество строк кода (LOC) для C++, Java и Python составляет в 2,5, 2, и в 1,5 раза больше соответственно, чем у Julia для той же модели [Sells, 2020].

Более того, из-за подхода к компиляции «точно в срок» (JIT — Just-In-Time), основанного на низкоуровневой виртуальной машине (LLVM), Julia немного медленнее, чем статические языки, но намного быстрее, чем другие динамические языки. Так, это наглядно демонстрируют результаты микротестов, которые проверяют производительность компилятора на ряде распространенных шаблонов кода, таких как вызовы функций, синтаксический анализ строк, сортировка, числовые циклы, генерация случайных чисел [Геворкян и др., 2022], рекурсия и операции с массивами [Julia Micro-Benchmarks…, 2023].

Кроме того, в недавней работе одного из лидеров в области развития Julia, К. Ракаукаса, представлено обзор решений дифференциальных уравнений в таких средах, как MATLAB, R, Julia, Python, C, Mathematica, Maple, and Fortran.

Он отмечает, что по сравнению с большинством других наборов, которые предлагают не более пятнадцати методов на высоком уровне, библиотека (пакет) DifferentialEquations.jl предлагает более двухсот методов, при этом функционал библиотеки постоянно расширяется [Rackauckas, 2023]. Так, как и стандартные пакеты Python и R, DifferentialEquations.jl предлагает обертки для методов Sundials, ODEPACK и Hairer. Однако, поскольку код Julia всегда компилируется в JIT, с точки зрения эффективности его оболочки больше похожи на PyDSTool или JiTCODE. Таким образом, DifferentialEquations.jl доступны все стандартные методы, имеющиеся в иных решениях.

В частности, для решения обычных дифференциальных уравнений библиотека DifferentialEquations.jl включает такие методы, как методы Рунге-Кутты, (E)SDIRK методы, методы Розенброка, а также некоторые уникальные методы (методы Вернера, метод Богацки-Шампина, методы OwrenZen и мн. др.). При этом и собственные методы Julia также очень хорошо тестируются, и все тесты находятся в открытом доступе. При этом пользо-

Таблица 2.1.

Сравнительный анализ программного обеспечения для решения дифференциальных уравнений

Тема/продукт	MATLAB	SciPy	deSolve	DifferentialEquations.jl	Sundials	Hairer	ODEPACK/Netlib /NAG
Язык	MATLAB	Python	R	Julia	C++ and Fortran	Fortran	Fortran
Выбор методов для ОДУ	2	1	1	4	3	3	3
Эффективность	1	1	1	4	4	3	3
Настраиваемость	2	1	1	4	4	3	3
Обработка событий	3	3	2	4	3	0	3
Символическое вычисление якобианов и автоматическое дифференцирование	0	0	0	3	0	0	0
Комплексные числа	4	3	0	3	0	0	0
Числа произвольной точности	0	0	0	4	0	0	0
Управление решателями	0	1	0	4	4	3	—
Встроенный параллелизм	0	0	0	4	4	0	0
Решатели дифференциально-алгебраических уравнений (DAE)	3	0	3	4	3	4	3
Неявно определенные решатели DAE	3	0	4	2	4	0	4
Решатели дифференциальных уравнений с постоянной задержкой (DDE)	2	0	1	4	0	3	2
Зависящие от состояния решатели DDE	1	0	1	4	0	4	3
Решатели стохастических дифференциальных уравнений (SDE)	1	0	0	4	0	0	0
Специализированные методы для ОДУ второго порядка и гамильтониана (и симплектических интеграторов)	0	0	0	4	0	3	0
Решатели краевых задач (BVP)	3	2	0	3	0	0	3
Совместимость с графическим процессором	0	0	0	4	0	0	0
Аналитические дополнения (анализ чувствительности, оценка параметров и т.д.)	0	0	0	4	4	0	3

Источник: [Rackauckas, 2023]

Таблица 2.2.

Сравнительный анализ программного обеспечения для решения дифференциальных уравнений

Тема/продукт	JitCODE	PyDSTool	FATODE	GSL	BOOST	Mathematica	Maple
Язык	Python	Python	Fortran	C	C++	Mathematica	Maple
Выбор методов для ОДУ	1	1	3	1	2	2	2
Эффективность	3	3	3	2	2	2	3
Настраиваемость	2	2	2	2	2	3	2
Обработка событий	0	2	0	0	0	3	3
Вычисление якобианов и авто-дифференцирование	0	0	0	0	0	4	4
Комплексные числа	0	0	0	0	3	4	4
Числа произвольной точности	0	0	0	0	4	4	4
Управление линейными/нелинейными решателями	0	0	0	0	0	2	0
Встроенный параллелизм	0	0	0	0	2	0	0
Решатели дифференциально-алгебраических уравнений (DAE)	0	2	3	0	0	3	3
Неявно определенные решатели DAE	0	0	0	0	0	3	0
Решатели дифференциальных уравнений с постоянной задержкой (DDE)	2	0	0	0	0	3	4
Зависящие от состояния решатели DDE	0	0	0	0	0	0	4
Решатели стохастических дифференциальных уравнений (SDE)	3	0	0	0	0	2	1
Специализированные методы для ОДУ второго порядка и гамильтониана (и симплектических интеграторов)	0	0	0	0	2	3	0
Решатели краевых задач (BVP)	0	0	0	0	0	3	2
Совместимость с графическим процессором	0	0	0	0	4	0	0
Аналитические дополнения (анализ чувствительности, оценка параметров и т.д.)	0	1	3	0	0	4	0

Источник: [Rackauckas, 2023]

вателю предоставляется полный контроль, поскольку сами решатели дифференциальных уравнений написаны, по сути, как метод в интерфейсе обработки событий, что означает, что все, что решатели могут делать внутри, может делать и пользователь (исследователь).

В таблицах (табл. 2.1–2.2) представлены сравнение библиотеки DifferentialEquations.jl с тринадцатью иными продуктами для решения дифференциальных уравнений. Функциональность каждого программного обеспечения определяется по четырёхбалльной шкале, где: 0 — функциональность не существует; 1 — функциональность существует, но является неполной; 2 — базовые функции существуют; 3 — базовые функции существуют и существует некоторая дополнительная возможность настройки (например, может включать дополнительные методы для повышения эффективности); 4 — продукт имеет все базовые функции и многое другое (например, дополнительные функции для гибкости и эффективности).

Сравнительный анализ (табл. 2.1–2.2) показывает, что в DifferentialEquations.jl, по сути, отражены сильные стороны каждого из других наборов. Главная причина этого заключается в том, что библиотека DifferentialEquations.jl изначально была разработана именно для этого.

Единственным недостатком является то, что из-за того, что DifferentialEquations.jl настолько ориентирован на функциональность и производительность, его документация является очень объемной [Rackauckas, 2023]. Однако, отметим, что, она представлена в максимально

открытой и явной форме, а это, предполагается, у заинтересованных учёных вызовет интерес, который перекроет вызванные трудности с чтением документации Julia и, в частности, DifferentialEquations.jl.

Заключение

Язык Julia — это относительно молодой язык, который, тем не менее, быстро внедряется в области науки о данных и научных вычислений благодаря той высокой производительности, которую он предлагает. Вследствие этого многим ученым, использующим Julia, эти инструменты понадобятся либо в качестве средства для анализа самих моделей, либо в качестве промежуточных инструментов в более сложных методах.

Благодаря своей применимости ко многим классам дифференциальных уравнений, включенным инструментам анализа для выполнения оценки параметров и анализа чувствительности, а также быстрому темпу разработки этого программного обеспечения, DifferentialEquations.jl выглядит жизнеспособным выбором для многих желающих исследователей, которые используют библиотеку дифференциальных уравнений.

Кроме того, Julia — это легко расширяемая среда, поскольку представляет собой программное обеспечение с открытым исходным кодом и модульной структурой. Язык Julia довольно конкурентоспособен для использования в научных вычислениях. JIT-компиляция, основанная на LLVM и специальной системе типов, позволяет Julia решать ресурсоемкие задачи. Более того, стандартные пакеты Julia могут превосходить по эффективности некоторые коммерческие программы, что является исключительным преимуществом решения дифференциальных уравнений числовыми методами.

ЛИТЕРАТУРА

1. Зырянов Д.М., Емельянов А.А., Лещенко К.С. Язык научных вычислений Julia // Оригинальные исследования. — 2020. — Т. 10. — №. 9. — С. 50–56.
2. Ильичев В.Ю., Юрик Е.А., Медов Д.С. Решение дифференциальных уравнений в частных производных с использованием функций языка Julia // E-Scio. — 2022. — №. 2 (65). — С. 251–260.
3. Геворкян М.Н., Королькова А.В., Кулябов Д.С. Реализация гиперболических комплексных чисел на языке Julia // Discrete and Continuous Models and Applied Computational Science. — 2022. — Т. 30. — №4. — С. 318–329.
4. Кулябов Д.С., Королькова А.В. Компьютерная алгебра на Julia // Программирование. — 2021. — №. 2. — С. 44–50.
5. Рожков А.В., Барсукова А.С. Экспериментальная математика и язык Julia–локальное распределение простых чисел // Новые информационные технологии в образовании и науке. — 2022. — №. 2. — С. 6.
6. Bezanson J. et al. Julia: A fast dynamic language for technical computing // arXiv preprint arXiv:1209.5145. — 2012.
7. Bezanson J. et al. Julia: A fresh approach to numerical computing // SIAM review. — 2017. — Т. 59. — №. 1. — pp. 65–98.
8. Hairer E., Lubich C., Wanner G. Numerical geometric integration // Unpublished Lecture Notes, March. — 1999.
9. Lauwens B., Downey A. B. Think Julia: how to think like a computer scientist. — O'Reilly Media, 2019.
10. Lindner M. et al. NetworkDynamics. jl—Composing and simulating complex networks in Julia // Chaos: An Interdisciplinary Journal of Nonlinear Science. — 2021. — Т. 31. — №. 6
11. Kwong T. Hands-On Design Patterns and Best Practices with Julia: Proven solutions to common problems in software design for Julia 1. x. — Packt Publishing Ltd, 2020.
12. Rackauckas C., Nie Q. Differentialequations.jl — a performant and feature-rich ecosystem for solving differential equations in Julia //Journal of open research software. — 2017. — Т. 5. — №. 1.
13. Rackauckas C. A comparison between differential equation solver suites in MATLAB, R, Julia, Python, C, Mathematica, Maple, and Fortran //Authorea Preprints. — 2023.
14. Xiao L. et al. Julia language in computational mechanics: A new competitor //Archives of Computational Methods in Engineering. — 2022. — Т. 29. — №. 3. — С. 1713–1726.
15. Sells R. Julia programming language benchmark using a flight simulation // 2020 IEEE Aerospace Conference. — IEEE, 2020. — pp. 1-8.
16. Simulating the Outer Solar System. — URL: https://docs.sciml.ai/DiffEqDocs/stable/examples/outer_solar_system/
17. Julia 1.9 Documentation. — URL: https://docs.julialang.org/en/v1/
18. Julia Micro-Benchmarks. — URL: https://julialang.org/benchmarks/

Журнал «Современная наука: актуальные проблемы теории и практики»