*ПРОГРАММИРОВАНИЕ, 2019, № 2, с. 41–48*

**КОМПЬЮТЕРНАЯ АЛГЕБРА**

*УДК 681.3.06 + 514.74*

# НОВЫЕ ВОЗМОЖНОСТИ ВТОРОЙ ВЕРСИИ ПАКЕТА КОМПЬЮТЕРНОЙ АЛГЕБРЫ CADABRA

**© 2019 г. Д. С. Кулябов[a,b,*], А. В. Королькова[a], Л. А. Севастьянов[a,c]**

[a] *Кафедра прикладной информатики и теории вероятностей, Российский университет дружбы народов 117198 Россия, Москва, ул. Микулохо-Маклая, 6*

[b] *Лаборатория информационных технологий, Объединенный институт ядерных исследований 141980 Россия, Московская область, Дубна, ул. Жолио-Кюри, 6*

[c] *Лаборатория теоретической физики, Объединенный институт ядерных исследований 141980 Россия, Московская область, Дубна, ул. Жолио-Кюри, 6*

**E-mail: kulyabov-ds@rudn.ru*

Поступила в редакцию 02.09.2018 г.
После доработки 02.09.2018 г.
Принята к публикации 20.08.2018 г.

В определенных научных областях есть потребность использования операций над тензорами. Для облегчения кропотливой работы над тензорными расчетами можно использовать системы компьютерной алгебры. В качестве основной системы тензорной компьютерной алгебры авторами данной работы в своих научных исследованиях уже несколько лет используется система Cadabra. Недавно вышла работоспособная вторая версия этой программы. В ней сделан ряд улучшений, которые можно позиционировать как революционные. Наиболее яркие улучшения касаются реализации компонентных тензорных операций и смены идеологии программной реализации системы по сравнению с первой версией Cadabra. В данной статье дается краткий обзор ключевых улучшений в системе Cadabra.

**DOI:** 10.1134/S0132347419020079

## I. ВВЕДЕНИЕ

Системы компьютерной алгебры с поддержкой операций над тензорами можно условно разделить на три группы [1]. К первой группе относят универсальные системы тензорных вычислений, которые создавались в основном для применения в задачах общей теории относительности. В таких системах упор делается на вычисление характерных величин римановой геометрии (символы Кристоффеля, тензор Римана) [2, 4]. Ко второй группе относят системы тензорной компьютерной алгебры для расчетов в квантовой теории поля [5, 9]. В них упор делается на вычисления с использованием дираковских спиноров, а также используются простые симметрии. Третья группа систем тензорной компьютерной алгебры создавалась скорее как фреймворк для произвольных тензорных расчетов. В них меньше готовых шаблонов, но есть более выразительные средства для конструирования новых объектов. Именно к этой группе и относится система Cadabra, особенностям реализации которой посвящена данная работа.

Система Cadabra на данный момент имеет две версии. В первой версии Cadabra (далее Cadabra1) есть существенные отличия от других систем тензорной компьютерной алгебры, в частности:

- использование $\TeX$-нотации при записи выражений;
- удобство определения произвольного типа тензоров;
- использование диаграмм Юнга для описания симметрийных свойств тензоров.

Тем не менее Cadabra1 имеет и существенные недостатки. В частности, в этой версии отсутствует возможность компонентных тензорных вычислений. Кроме того, монолитная и косная программная структура самой системы не дает возможности надеяться на реализацию компонентных вычислений в ближайшем будущем. Однако, революционный шаг, совершенный при реализации второй версии системы (далее Cadabra2), а именно использование экосистемы Python, позволило решить указанные проблемы. На наш взгляд именно комбинация “компонентные вычисления + экосистема Python”, реализованная во второй версии, является революционным улучшением системы Cadabra.

К сожалению, документация по системе Cadabra оставляет желать лучшего. На сайте самой системы (https://cadabra.science/) представлен достаточно небольшой набор примеров. Более существенную информацию можно получить из специализированных исследовательских статей, в которых можно встретить возможное применение системы Cadabra [10, 15].

В данной работе предлагается небольшой обзор новых возможностей второй версии пакета компьютерной алгебры Cadabra. Структура статьи следующая. В разделе II рассмотрены особенности реализации первой и второй версий Cadabra. В разделе III приведено описание процесса варьирования действия для электромагнитного поля без источников с целью сделать краткий обзор синтаксиса Cadabra2. Если читатель хочет сравнить синтаксис Cadabra2 с синтаксисом Cadabra1, то мы рекомендуем ознакомиться с более ранними работами [13, 15]. В разделе IV рассмотрена одна из важнейших особенностей Cadabra2 — прозрачное взаимодействие с универсальной скалярной системой компьютерной алгебры SymPy. В разделе V описывается основное, на наш взгляд, нововведение в Cadabra2 — компонентные вычисления, рассмотренные на примере нахождения основных величин для общей теории относительности, а именно связности Кристоффеля и разных кривизн. В завершение статьи, в разделе VI приводится пример, демонстрирующий работу с графикой в Cadabra2.

Заметим, что при реализации примеров были использованы элементы кода из документации по Cadabra2 (https://cadabra.science/tutorials.html).

## II. ОСОБЕННОСТИ РЕАЛИЗАЦИИ CADABRA1 И CADABRA2

Каждая система компьютерной алгебры имеет свою специфику в реализации. Можно выделить несколько уровней при реализации:

- нотация;
- язык манипуляции;
- язык реализации;
- язык расширений.

Не у каждой системы компьютерной алгебры можно выделить все эти уровни. Так, у большинства систем нотация строится на основе языка манипуляции.

У пакета Cadabra2, также как и у пакета Cadabra1 нотация построена на основе $\TeX$-нотации (правильнее сказать, $\TeX$-подобной нотации). Для этого выделяется некоторое подмножество символов $\TeX$, как то буквы разных алфавитов, символы производных, интеграла и так далее. Главное, что индексы обозначаются с помощью символов _ и ^, как привычно пользователям системы $\TeX$.

Язык манипуляций служит собственно для работы в системе. Синтаксис Cadabra1 достаточно простой и скорее приспособлен для облегчения лексического и синтаксического разбора текста программы, чем для удобства пользователя. Такой подход к конструированию языков использовали в ранние годы развития вычислительной техники (например, в языках Shell, Perl). В Cadabra2 произошел качественный скачок — система перешла на язык Python. Соответственно и все операции в Cadabra2 выполняются в Python-подобном синтаксисе.

Реализация системы Cadabra1 выполнена на языке C++ с использованием системы компьютерной алгебры LiE [16] (на данный момент компиляция системы LiE вызывает некоторые трудности). Основное предназначение — работа с группами Ли, на которых основаны операции с симметриями тензоров. Следует заметить, что фактически вся система была реализована одним человеком. Это большой труд. Но наличие только одного автора и монолитная программная структура системы вызвала тревогу за дальнейшую судьбу всей системы.

В версии Cadabra2 автор принципиально изменил свой подход к структуре системы. Он взял направление на интеграцию ее с экосистемой Python. Система все также написана на языке C++, но в качестве языка-клея применяется Python, который также используется в качестве языка манипуляций. При этом Python же может быть использован как язык написания расширений.

Инфраструктура Python открывает доступ к огромному количеству научных библиотек, в том числе и к проекту SciPy [17]. Это позволяет использовать библиотеки SciPy бесшовно, прозрачно для пользователя. Именно это и позволило на наш взгляд сделать системе Cadabra2 революционный шаг.

## III. ЭЛЕМЕНТЫ СИНТАКСИСА CADABRA2

Напомним некоторые элементы синтаксиса Cadabra1 и Cadabra2. В качестве примера рассмотрим получение уравнения Максвелла без источников [18]:

$$\partial_{\mu} F^{\mu\nu} = 0, \tag{1}$$

варьируя действие

$$S = -\frac{1}{4}\int F_{\mu\nu} F^{\mu\nu} x. \tag{2}$$

При этом тензор Максвелла $F_{\mu\nu}$ представим через векторный потенциал:

$$F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu. \tag{3}$$

Напомним, что обе версии Cadabra используют нотацию $\TeX$. Во второй версии появилась возможность использовать функционал языка Python, в частности, можно задавать функции прямо в тексте программы, написанной на Cadabra2. Заметим, что также как и Cadabra1, Cadabra2 по умолчанию практически не обрабатывает получающиеся выражения (разве что приводит подобные), эта обработка отдана на откуп самому пользователю. Если требуется применить какой-то набор правил ко всем используемым выражениям, то можно задать функцию `post_process`, выполняемую после каждой операции (фактически, `post_process` является просто функцией Python):

```
def post_process (ex):
    sort_product (ex)
    canonicalise (ex)
    collect_terms (ex)
```

В экстремальном случае эту функцию можно сделать пустой, убрав таким образом любую обработку выражений.

Интерфейс Cadabra2 выполнен по идеологии блокнота, то есть наряду с программным кодом можно указывать и пояснения. Правда, в отличие от iPython [19] текстовая часть пишется не в синтаксисе Markdown [20], а в синтаксисе $\LaTeX$.

Объекту Cadabra2 можно приписать свойство, имеющее в свою очередь некоторый набор настроек. Естественно, раз речь идет о тензорах, наиболее полезное свойство — `Indices`. Настройка `position=free` дает возможность системе самой поднимать и опускать индексы:

```
{\mu, \nu, \rho} :: Indices (position=free).
x :: Coordinate.
\partial {#} :: Derivative.
```

Здесь знак # является шаблоном подстановки. Знак точки в конце выражения подавляет вывод (как часто принято в системах компьютерной алгебры).

Для работы с абстрактными индексами необходимо учитывать симметрийные свойства тензоров. Кроме того, при дифференцировании и интегрировании необходимо учитывать зависимость объектов от координат:

```
F_{\mu\nu} :: AntiSymmetric;
F_{\mu\nu} :: Depends (x).
A_{\mu} :: Depends (x, \partial{#}).
\delta{#} :: Accent;
```

Attached property AntiSymmetric to $F_{\mu\nu}$.

Attached property Accent to $\delta\#$.

В данном случае знак вариации $\delta$ рассматривается как модификатор, а не как объект со свойством `Derivative`, и не вносит никакой дополнительной вычислительной семантики.

В рассматриваемом нами примере $F_{\mu\nu}$ является тензором Максвелла. Распишем его через векторный потенциал $A_\mu$ (3):

```
F: = F_{\mu\nu} = \partical_{mu}{A_\nu}} - \partial_{\nu}{A_{mu}};
```

$$F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu.$$

Здесь знаком := задается метка строки (в нашем случае это $F$). Метка именует выражение для удобства ссылки на него в последствии.

Далее зададим действие для электромагнитного поля (2):

```
S:= -1/4 \int{F_{\mu\nu} F^{\mu\nu}}{x}
```

$$-\frac{1}{4}\int F^{\mu\nu}F_{\mu\nu}dx.$$

Путем подстановки перепишем действие через векторный потенциал $A_\mu$:

```
substitute (S, F);
```

$$-\frac{1}{4}\int(\partial_\mu A_\nu - \partial_\nu A_\mu)(\partial^\mu A^\nu - \partial^\nu A^\mu)dx.$$

После этого посмотрим вид действия:

```
S;
```

$$-\frac{1}{4}\int(\partial_\mu A_\nu - \partial_\nu A_\mu)(\partial^\mu A^\nu - \partial^\nu A^\mu)dx.$$

Значение действия в этом выражении не такое, какое мы задали изначально. Оно стало таким, каким мы его получили после последних вычислений. И это немного необычно. Дело в том, что большинство систем компьютерной алгебры реализованы на функциональных языках программирования или придерживаются функциональной парадигмы, в рамках которой переменные обладают свойством иммутабельности. В данном же случае метка ведет себя как переменная в императивных языках программирования (собственно, Python и является императивным языком). Это делает работу в Cadabra2 необходимо линейной: нельзя произвольно передвигаться по блокноту и производить вычисления в произвольном месте.

Проварьируем действие:

```
vary(S, $A_{\mu}->\delta{A_{mu}}$);
```

$$-\frac{1}{4}\int((\partial^{\mu}A^{\nu}-\partial^{\nu}A^{\mu})(\partial_{\mu}\delta A_{\nu}-\partial_{\nu}\delta A_{\mu})$$
$$+(\partial_{\mu}A_{\nu}-\partial_{\nu}A_{\mu})(\partial^{\mu}\delta A^{\nu}-\partial^{\nu}\delta A^{\mu}))dx.$$

Здесь мы видим пример использования не метки, а непосредственно выражения. Для этого мы окружаем выражение символами доллара ($), как делается в обычном $\TeX$'е.

Далее раскроем произведения и приведем подобные:

```
distribute (S);
```

$$-\frac{1}{4}\int(4\partial^{\mu}A^{\nu}\partial_{\mu}\delta A_{\nu}-4\partial^{\mu}A^{\nu}\partial_{\nu}\delta A_{\mu})dx.$$

Затем проинтегрируем по частям:

```
integrate_by_parts(S,   $\delta{A_{\mu}}$);
```

$$-\frac{1}{4}\int(-4\delta A^{\mu}\partial^{\nu}(\partial_{\nu}A_{\mu})+4\delta A^{\mu}\partial^{\nu}(\partial_{\mu}A_{\nu}))dx.$$

Интегрирование по частям в данном случае является достаточно формальным действием, использующим только свойство Derivative объекта.

Еще раз выполним подстановку, а затем раскроем произведения и приведем подобные:

```
substitute  (_,  $\partial_{\mu}{A_{\nu}}  ->  1/2\partial_{\mu}{A_{\nu}} + ↪1/2 F_{\mu\nu} + 1/2 \partial_{\nu}{A_{\mu}}$);
```

$$-\frac{1}{4}\int\left(-4\delta A^{\mu}\partial^{\nu}\left(\frac{1}{2}\partial_{\nu}A_{\mu}-\frac{1}{2}F_{\mu\nu}+\frac{1}{2}\partial_{\mu}A_{\nu}\right)\right.$$
$$\left.+4\delta A^{\mu}\partial^{\nu}\left(\frac{1}{2}\partial_{\mu}A_{\nu}+\frac{1}{2}F_{\mu\nu}+\frac{1}{2}\partial_{\nu}A_{\mu}\right)\mathrm{d}x\right.$$

```
distribute (_);
```

$$-\int\delta A^{\mu}\partial^{\nu}F_{\mu\nu}dx.$$

Здесь специальная метка _ указывает на предыдущее выражение.

В результате получим искомое уравнение Максвелла (1):

$$\partial^{\nu}F_{\mu\nu}=0.$$

Таким образом показано, что синтаксис языка манипуляции выражениями в системе Cadabra2 базируется на синтаксисе языка Python и он более привычен, чем синтаксис языка Cadabra1.

## IV. ВЗАИМОДЕЙСТВИЕ CADABRA С SYMPY

Тензорные системы компьютерной алгебры имеют в своем арсенале достаточно небольшое число поддерживаемых операций. Их достаточно для основных манипуляций с тензорами в формализме абстрактных индексов и в безындексном формализме. Но для многих операций (например, для полноценной реализации компонентных вычислений) требуется возможность использования скалярных операций. Если тензорная система компьютерной алгебры реализована в рамках универсальной системы компьютерной алгебры, то проблем не возникает. Однако Cadabra является отдельной системой. В Cadabra1 реализован, хотя и неудобный в применении, механизм для связи с универсальной системой компьютерной алгебры Maxima – создалось впечатление, что этот механизм реализован только как "доказательство концепции".

В Cadabra2 связь с универсальной системой компьютерной алгебры реализована через SymPy [21]. Причем эта связь бесшовная: для пользователя использование данного механизма проходит незаметно. Впрочем, это один из результатов реализации системы Cadabra2 на языке Python.

Продемонстрируем применение SymPy в Cadabra2 на примере, в котором нам необходимо вычислить интеграл:

$$\int\frac{1}{x}\mathrm{d}x.$$

Основная функция явного вызова SymPy – `map_sympy()`. Эта функция имеет побочное действие: она изменяет значение аргумента. Впрочем, как мы уже обсуждали, отсутствие иммутабельности является особенностью системы Cadabra2. Рассмотрим простейший вызов этой функции:

```
ex := \int{1/x}{x};
```

$$\int x^{-1}dx$$

```
map_sympy(_);
```

$$\log(x).$$

Для того, чтобы убедиться в наличии побочного действия, посмотрим текущее значение выражения `ex`:

```
ex;
```

$$\log(x).$$

Мы увидим, что значение `ex` изменилось.

Можно передать значение выражения не просто в систему SymPy, а вызвать для его обработки конкретную функцию. Например, в рассматриваемом нами случае можно вызвать функцию `integrate` среды SymPy:

```
ex := 1/x;
```

$$x^{-1}.$$

Вторым аргументом функции `map_sympy()` идёт имя конкретной функции SymPy:

```
map_sympy(ex, "integrate");
```

$$\log(x)$$

```
ex;
```

$$\log(x).$$

Здесь еще раз убеждаемся в наличии побочного действия функции `map_sympy()`.

В рамках языка Python есть несколько вариантов выполнения одного и того же действия. Естественно, эта возможность есть и в Cadabra2. Рассмотрим следующие варианты исключительно как потенциальные возможности, а не как руководство к действию.

Возможно использование метода класса `_sympy_()`:

```
ex := \int{1/x}{x};
```

$$\int x^{-1}\mathrm{d}x.$$

Воспринимая метку ex как объект, вызовем метод `_sympy_()`:

```
ex. _sympy_();
```

$$\log(x)$$

Проверим состояние среды:

```
ex;
```

$$\int x^{-1}\mathrm{d}x.$$

Видим, что состояние среды не изменилось, то есть метод класса `_sympy_()` не имеет побочного действия.

Кроме того, можно использовать функцию `sympy` с методом, соответствующим вызываемой функции среды SymPy:

```
ex := 1/x;
```

$$x^{-1}.$$

Вызовем функцию `sympy` с методом `integrate`:

```
sympy.integrate(ex);
```

$$\log(x).$$

Обратим внимание, что данная функция всегда требует указания конкретного метода, поэтому мы не смогли бы применить ее в предыдущем случае.

Опять проверим состояние среды:

```
ex;
```

$$x^{-1}.$$

Убеждаемся, что функция `sympy` не обладает побочным действием.

Базируясь на приведенных выше примерах, можно сделать вывод, что взаимодействие с SymPy в Cadabra2 реализовано достаточно элегантным образом. Но основное достоинство этой операции заключается на наш взгляд в ее тесной интеграции с системой, например, для реализации компонентных вычислений (см. раздел V).

## V. КОМПОНЕНТНЫЕ ВЫЧИСЛЕНИЯ В CADABRA2

В качестве примера компонентных тензорных операций получим кривизну $R$ на сфере $S^2$ радиуса $r$:

$$g_{\alpha\beta} = \mathrm{diag}(r^2, r^2\sin^2\vartheta).$$

Для этого получим значения для символов Кристоффеля $\Gamma^{\alpha}_{\mu\nu}$, тензора Римана $R^{\alpha}_{\beta\mu\nu}$ и тензора Риччи $R_{\alpha\beta}$ [18, 22].

Зададим координаты и набор меток для индексов. При этом укажем, какие значения данные метки принимают:

```
{\theta, \varphi} :: Coordinate;
{\alpha, \beta, \gamma, \delta, \rho, \sigma, \mu, \nu, \lambda} ::
    ↪ Indices (values={\varphi, \theta}, position=fixed};
\partial{#} :: PartialDerivative;
```

Attached property Coordinate to $[\theta, \varphi]$.

Attached property Indices (position fixed) to $[\alpha, \beta, \gamma, \delta, \rho, \sigma, \mu, \nu, \lambda]$.

Attached property PartialDerivative to $\partial\#$.

Затем зададим тензор $g$ со свойством `Metric` и аналогичным образом зададим обратную метрику:

```
g_{\alpha\beta} :: Metric.
g^{\alpha\beta} :: InverseMetric.
```

В данном случае достаточно задать компоненты для самой метрики. Компоненты для обратной метрики рассчитываются с помощью функции `complete`:

```
g:={ g_{\theta\theta} = r**2, g_{\varphi\varphi} = r**2 \sin(\theta)**2
↪ }.
complete (g, $g^{alpha\beta}$);
```

$$[g_{\theta\theta} = r^2, g_{\varphi\varphi} = r^2(\sin\theta)^2, g^{\varphi\varphi} = (r^2(\sin\theta)^2)^{-1}, g^{\theta\theta} - r^{-2}]$$

Запишем определение символов Кристоффеля $\Gamma^{\alpha}_{\mu\nu}$:

```
Gamma:= \Gamma^{\alpha}_{\mu\nu} = 1/2g^{\alpha\beta} (\partial)_{\nu}{
```

```
  ↪ g_\beta/mu}}   +   \partial_{\mu}
{g_{\beta\nu}} - \partial_{\beta}}g_
  ↪ {\mu\nu}});
```

$$\Gamma^{\alpha}_{\mu\nu} = \frac{1}{2} g^{\alpha\beta}(\partial_{\nu} g_{\beta\mu} + \partial_{\mu} g_{\beta\nu} - \partial_{\beta} g_{\mu\nu}).$$

Раскроем символы Кристоффеля через метрический тензор. При этом раскрытие будет применяться только к правой части определения (после знака равенства). Этот порядок раскрытия определяется опцией `rhsonly=True`. Функция `evaluate()` имплицитно использует вызов SymPy для операций с компонентами (это еще одно преимущество использования инфраструктуры Python). Для раскрытия тригонометрических соотношений следует явно обратиться к SymPy:

```
evaluate(Gamma, g, rhsonly=True)
map_sympy (Gamma, "expand_trig");
```

$$\Gamma^{\alpha}_{\mu\nu} = \square^{\alpha}_{\mu\nu}\begin{cases}\square^{\varphi}_{\varphi\theta} = (\tan\theta)^{-1}\\ \square^{\varphi}_{\varphi\theta} = (\tan\theta)^{-1}\\ \square^{\theta}_{\varphi\varphi} = -\sin\theta\cos\theta.\end{cases}$$

Аналогично для тензора Римана $R^{\alpha}_{\beta\mu\nu}$ запишем его определение и вычислим его компоненты:

```
R4:=R^{\rho}_{\sigma\mu\nu}   =   +
\partial_{mu}{\Gamma^{\rho}_{\sigma\nu
  ↪}} - \partial_{\nu}{\Gamma^{\rho}
_{\sigma\mu}} + \Gamma^{rho}_{\
  ↪beta\mu}\Gamma^{\beta}_{\sigma\
nu} - \Gamma^{\rho}_{\beta\nu} \
  ↪ Gamma^{\beta}_{\sigma\mu};
substitute (R4, Gamma)
evaluate (R4, g, rhsonly=True);
```

$$R^{\rho}_{\sigma\mu\nu} = \partial_{\mu}\Gamma^{\rho}_{\sigma\nu} - \partial_{\nu}\Gamma^{\rho}_{\sigma\mu} + \Gamma^{\rho}_{\beta\mu}\Gamma^{\beta}_{\sigma\nu} - \Gamma^{\rho}_{\beta\nu}\Gamma^{\beta}_{\sigma\mu}$$

$$R^{\rho}_{\sigma\mu\nu} = \square^{\rho}_{\sigma\nu\mu}\begin{cases}\square^{\theta}_{\varphi\varphi\theta} = (\sin\theta)^{2}\\ \square^{\varphi}_{\theta\varphi\theta} = -1\\ \square^{\theta}_{\varphi\theta\varphi} = -(\sin\theta)^{2}\\ \square^{\varphi}_{\theta\theta\varphi} = 1.\end{cases}$$

Тензор Риччи $R_{\alpha\beta}$ вычисляется из тензора Римана:

```
R2:=   R_{\sigma\nu}   =   R^{\rho}_
{\sigma\rho\nu};
substitute (R2, R4)
evaluate (R2, g, rhsonly=True);
```

$$R_{\sigma\nu} = R^{\rho}_{\sigma\rho\nu},$$

$$R_{\sigma\nu} = \square_{\sigma\nu}\begin{cases}\square_{\varphi\varphi} = (\sin\theta)^{2}\\ \square_{\theta\theta} = 1.\end{cases}$$

И, наконец, вычислим скалярную кривизну $R$:

```
R:= R = R_{\sigma\nu}g^{\sigma\nu};
substitute (R, R2)
evaluate (R, g, rhsonly=True);
```

$$R = R_{\sigma\nu} g^{\sigma\nu},$$

$$R = 2r^{-2}.$$

Таким образом, наличие компонентных вычислений для пользователя выглядит в Cadabra2 как дополнительное свойство `Coordinate` и несколько функций, основной из которых является функция `evaluate()`, которая собственно вычисляет значения компонент.

На уровне системы компонентные вычисления реализованы в Cadabra2 через взаимодействие со скалярной системой компьютерной алгебры, а именно с SymPy.

## VI. ИСПОЛЬЗОВАНИЕ ГРАФИКИ В CADABRA2

Необходимость наличия графики в системе тензорной компьютерной алгебры вызывает много вопросов. Коротко говоря, по мнению авторов, она просто не нужна. Однако в данном случае наличие графических возможностей — не более, чем дополнительный (в чем-то побочный) эффект реализации Cadabra2 в рамках экосистемы Python. Поэтому и построение графиков в Cadabra2 не отличается от построения графиков в Python.

Сначала нужно выбрать библиотеку для построения графиков. В следующем примере использована популярная библиотека Matplotlib [23, 25]. Для численных расчетов использована библиотека NumPy [26, 27].

Подключим модули для `matplotlib` и `numpy`:

```
import matplotlib.pyplot as plt
import numpy as np
```

Построим векторное поле, то есть каждой точке пространства (в нашем случае плоскости) поставим в соответствие вектор с началом в этой точке. Пусть вектор $f$ имеет вид:

$$f^{i}(x,y) = \begin{pmatrix}\sin x\cos x\\ \cos y\end{pmatrix}. \tag{4}$$

Зададим квадратную сетку, на которой и будем вычислять значения векторного поля. Поскольку

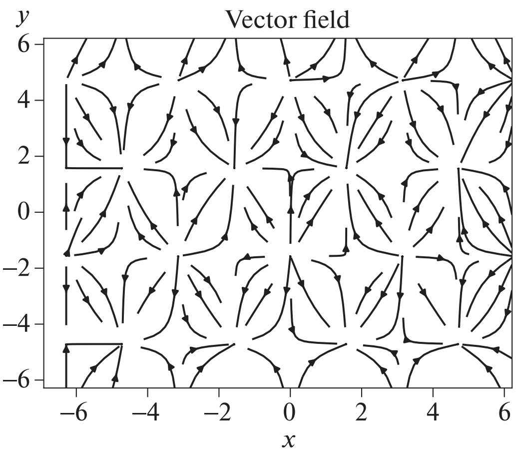


**Рис. 1.** Изображение векторного поля (4), полученное с помощью matplotlib.

отсчеты по осям одинаковы, то будем использовать только значения для *x*:

```
x = np.arange(-2*np.pi, 2*np.pi, 0.1)
u = np.sin(x)*np.cos(x)
v = np.cos(x)
uu, vv = np.meshgrid(u,v)
```

Функция `meshgrid` из пакета `numpy` создает прямоугольную сетку из двух массивов (в нашем случае из *u* и *v*).

Теперь, собственно, построим векторное поле с помощью функции `streamplot` из пакета `matplotlib`:

```
fig = plt.figure()
plt.streamplot(x, x, uu, vv, col-
or='black')
plt.title('Vector field')
plt.xlabel('x')
plt.ylabel('y')
```

Мы можем как сохранить получившийся график, так и отобразить его в рабочем блокноте:

```
fig.savefig("plot.pdf")
display(fig)
```

В результате получим изображение векторного поля (см. рис. 1).

Возможно в будущем в Cadabra появятся более полезные приложения, нежели мы продемонстрировали в этом простом примере.

## VII. ЗАКЛЮЧЕНИЕ

В результате можно сделать следующие выводы. Основным прорывом системы Cadabra2 с точки зрения пользователя тензорной системы компьютерной алгебры является реализация компонентных тензорных вычислений. Таким образом система охватывает весь спектр необходимых тензорных операций. С точки же зрения разработчика основным нововведением является переписывание системы с использованием языка Python и всей экосистемы этого языка. Это позволяет надеяться на рост интереса к системе Cadabra2 при решении задач, связанных с операциями над тензорами.

Публикация подготовлена при поддержке Программы РУДН "5-100" и при финансовой поддержке РФФИ в рамках научных проектов № 16-07-00556, 18-07-00567, 18-51-18005.

## СПИСОК ЛИТЕРАТУРЫ

1. *MacCallum M.A.H.* Computer algebra in gravity research // \BibEmph{LivingReviewsinRelativity}. 2018. V. 21. № 1. P. 1–93.
2. I*lyn V., Kryukov A.* ATENSOR – REDUCE program for tensor simplification // \BibEmph{ComputerPhysicsCommunications}. 1996. V. 96. № 1. P. 36–52.
3. *Gómez-Lobo A.G.P., Martín-Garcia J.M.* Spinors: A Mathematica package for doing spinor calculus in General Relativity // \BibEmph{ComputerPhysicsCommunications}. 2012. V. 183. № 10. P. 2214–2225. arXiv: 1110.2662.
4. *MacCallum M.* Computer Algebra in General Relativity // \BibEmph{InternationalJournalofModernPhysicsA}. 2002. V. 17. № 20. P. 2707–2710.
5. *Bolotin D.A., Poslavsky S.V.* Introduction to Redberry: the Computer Algebra System Designed for Tensor Manipulation. 2015. P. 1–27. arXiv: 1302.1219.
6. *Poslavsky S., Bolotin D.* Redberry: a computer algebra system designed for tensor manipulation // \BibEmph {JournalofPhysics: ConferenceSeries}. 2015. V. 608. № 1. P. 012060. arXiv: 1302.1219.
7. *Fliegner D., Retery A., Vermaseren J. a. M.* Parallelizing the Symbolic Manipulation Program FORM Part I: Worstation Clusters and Message Passing. 2000. arXiv: hep-ph/0007221.
8. *Heck A.* FORM for Pedestrians. 2000.
9. *Tung M.M.* FORM Matters: Fast Symbolic Computation under UNIX // \BibEmph{ComputersandMathematicswithApplications}. 2005. V. 49. № 7–8. P. 1127–1137. arXiv: cs/0409048.
10. *Peeters K.* Introducing Cadabra: a symbolic computer algebra system for field theory problems. 2007. arXiv: hep-th/0701238.
11. *Peeters K.* Cadabra: a field-theory motivated symbolic computer algebra system // \BibEmph{ComputerPhysicsCommunications}. 2007. V. 176. № 8. P. 550–558. arXiv: cs/0608005.
12. *Brewin L.* A Brief Introduction to Cadabra: A Tool for Tensor Computations in General Relativity // \BibEmph{ComputerPhysicsCommunications}. 2010. V. 181. № 3. P. 489–498. arXiv: 0903.2085.
13. *Sevastianov L.A., Kulyabov D.S., Kokotchikova M.G.* An Application of Computer Algebra System Cadabra to Scientific Problems of Physics // \BibEmph{PhysicsofParticlesandNucleiLetters}. 2009. V. 6. № 7. P. 530–534.
14. *Korol'kova A.V., Kulyabov D.S., Sevast'yanov L.A.* Tensor Computations in Computer Algebra Systems // \

BibEmph{ProgrammingandComputerSoftware}. 2013. V. 39. № 3. P. 135–142. arXiv: 1402.6635.

15. *Kulyabov D.S.* Using two Types of Computer Algebra Systems to Solve Maxwell Optics Problems // \BibEmph{ProgrammingandComputerSoftware}. 2016. V. 42. № 2. P. 77–83. arXiv: 1605.00832.
16. *Leeuwen M.A.A. v., Cohen A.M., Lisser B.* LiE: A package for Lie group computations. Amsterdam: Computer Algebra Nederland, 1992.
17. *Oliphant T.E.* Python for Scientific Computing // \BibEmph{ComputinginScienceandEngineering}. 2007. V. 9. № 3. P. 10–20.
18. *Ландау Л.Д., Лифшиц Е.М.* Теория поля. Теоретическая физика. Т. II. 8-е изд. М.: Физматлит, 2012. 536 с.
19. *Perea F., Granger B.E.* IPython: A System for Interactive Scientific Computing // \BibEmph{ComputinginScienceandEngineering}. 2007. V. 9. № 3. P. 21–29.
20. {Thetest/markdownMediaType}: RFC/RFC Editor; Executor: *S. Leonard*: 2016.
21. *Lamy R.* Instant SymPy Starter. Packt Publishing, 2013. 52 p.
22. *Мизнер Ч., Торн К., Уилер Дж.* Гравитация. Мир изд. М., 1977. Т. 1. 474 с.
23. *Tosi S.* Matplotlib for Python Developers. Packt Publishing, 2009. 308 p.
24. *Vaingast S.* Beginning Python visualization: crafting visual transformation scripts. Springer, 2009. 384 p.
25. *Müller A.C., Guido S.* Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly Media, 2016. 285 p.
26. *Idris I.* NumPy Cookbook. Packt Publishing, 2012. 226 p.
27. *Oliphant T.E.* Guide to NumPy. 2 edition. CreateSpace Independent Publishing Platform, 2015. 364 p.