Н. М. Ершов

Практическое введение в решение дифференциальных уравнений в Python

ДМК Пресс
издательство

Н. М. Ершов

Практическое введение в решение дифференциальных уравнений в Python

ДМК Пресс
издательство
Москва, 2022

УДК 517.9
ББК 22.161.61
Е80

Ершов Н. М.
Е80 Практическое введение в решение дифференциальных уравнений в Python. – М.: ДМК Пресс, 2022. – 176 с.: ил.

ISBN 978-5-93700-147-4

Книга посвящена вопросам практического применения символьных вычислений для решения различных прикладных задач, приводящих к дифференциальным уравнениям и их системам, с использованием библиотеки символьных вычислений SymPy языка Python.

Издание ориентировано на школьников старших классов, студентов, преподавателей и всех, интересующихся тематикой математического моделирования. Может служить дополнением к классическим учебникам по теории обыкновенных дифференциальных уравнений.

УДК 517.9
ББК 22.161.61

ISBN 978-5-93700-147-4

Оглавление

Введение

Настоящая книга посвящена вопросам практического применения символьных вычислений для решения различных прикладных задач, приводящих к дифференциальным уравнениям и их системам. Как известно, решение дифференциальных уравнений — с одной стороны, процесс многогранный и местами даже творческий, с другой — связан с выполнением больших объемов и чисто рутинной работы: арифметика, алгебраические преобразования, вычисления производных и интегралов и т. д. Такая деятельность является по большому счету чисто механической, поэтому она относительно легко и эффективно алгоритмизуется и может быть реализована программно на любом современном языке программирования.

Программные системы, позволяющие пользователю работать с математическими формулами, выполняя над ними те или иные символьные преобразования, называются системами *символьных вычислений*, или системами *компьютерной алгебры*. Первые такие системы появились еще в 60-х годах прошлого века. В настоящее время пользователям доступны десятки подобного рода систем — от коммерческих (Maple, Mathematica) до систем с открытым исходным кодом (Reduce, Maxima), некоторые из которых способны работать уже и на мобильных устройствах.

Система SymPy, которой и посвящена данная книга, является по сути обычной *библиотекой* языка Python. Такой подход к построению систем символьных вычислений имеет ряд преимуществ. Во-первых, система оказывается открытой и доступной всем без исключения пользователям. Во-вторых, работа с библиотекой SymPy в среде Jupyter Notebook позволяет проводить символьные вычисления прямо в браузере либо локально, либо удаленно с помощью какого-нибудь об-

РИС. 1 SymPy — библиотека Python для символьных вычислений

лачного сервиса[1], например Google Colab. В-третьих, символьные вычисления в рамках данной библиотеки оказываются интегрированными в обычную программу на языке Python, при этом пользователю системы оказывается доступным весь функционал языка Python, а также вся его инфраструктура в виде бесчисленного набора разнообразнейших пакетов и библиотек.

Предлагаемая читателю книга устроена достаточно просто. Каждая ее глава посвящена рассмотрению одной прикладной модели из физики, химии, биологии и т. д. После теоретического рассмотрения модели и вывода соответствующего ей дифференциального уравнения максимально детально описывается процесс формализации модели и решения возникающих в ней дифференциальных уравнений с помощью библиотеки SymPy[2]. Каждая глава сопровождается набором упражнений как технического характера — посчитать интеграл, решить уравнение, построить график, так и исследовательского — построить и исследовать по описанной схеме аналогичную модель.

Автор с благодарностью примет все конструктивные отзывы, комментарии и замечания относительно структуры и содержания настоящего издания, которые читатель может оставить или в телеграм-чате https://t.me/odesinsympy, или прислать автору на электронную почту по адресу ershovnm@gmail.com.

[1] При таком сценарии пользователю вообще не нужно устанавливать локально никакое программное обеспечение.

[2] Заметим, что автором не ставилась цель дать полное описание библиотеки SymPy. Такая задача, с одной стороны, является просто неподъемной, библиотека SymPy состоит из большого числа модулей, только часть из которых посвящена решению дифференциальных уравнений. Кроме того, данная библиотека все еще активно развивается, ее функционал постоянно расширяется и модифицируется. Более полную и актуальную информацию по работе с библиотекой SymPy читателю рекомендуется находить либо на официальном сайте библиотеки https://www.sympy.org, либо на профильных ресурсах в сети Интернет, например на сайте https://stackoverflow.com.

ГЛАВА 1

Вращение жидкости

Задача • В сосуд, имеющий форму прямого кругового цилиндра, налита жидкость, например вода. Сосуд вращается с постоянной угловой скоростью ω относительно оси цилиндра (рис. 1). Требуется определить, какую форму примет поверхность жидкости, если вращение продолжается достаточно долго[1]. При построении модели мы будем предполагать, что сосуд достаточно широкий и глубокий, это позволит пренебречь разными поверхностными эффектами около боковых стенок сосуда.

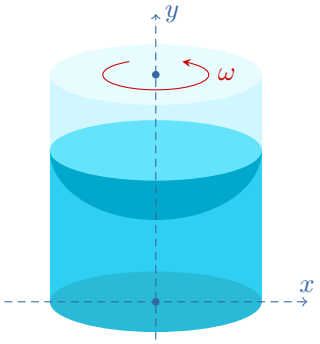


РИС. 1 Вращение сосуда с жидкостью

Очевидно, что искомая поверхность должна быть поверхностью вращения. Поэтому для нахождения ее формы достаточно рассмотреть осевое сечение нашего сосуда и найти форму соответствующей *кривой*, из которой потом мы сможем сформировать и саму поверхность.

Введем в модель систему координат, как это показано на рис. 1. Ось Oy направим по оси цилиндра, ось Ox — перпендикулярно оси Oy вдоль основания цилиндра, начало координат, таким образом, оказывается в центре основания цилиндра.

Рассмотрим малый объем ν жидкости на ее поверхности (точка P на рис. 2). Пусть его масса равна m. На этот объем действуют две силы: сила тяжести $F = mg$ и сила реакции опоры N со стороны всей остальной жидкости. Сила тяжести, как обычно, действует вертикально вниз, а реакция опоры — по нормали к нашей искомой кривой, т. е. перпендикулярно касательной к этой кривой в точке P.

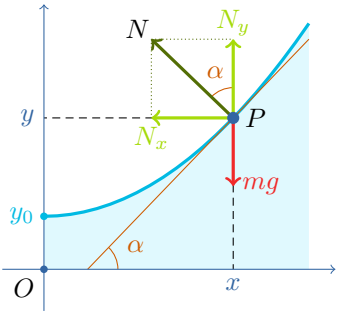


РИС. 2 Силы, действующие на малый объем жидкости ν массы m в точке $P(x, y)$

Обозначим угол наклона данной касательной к оси Ox через α и разложим силу N на горизонтальную N_x

[1] «Достаточно долго» понимается здесь в том смысле, что вся жидкость в сосуде должна прийти в стационарное состояние относительно самого сосуда, т. е. каждый элементарный ее объем будет совершать только общее вращательное движение с заданной угловой скоростью ω.

и вертикальную N_y составляющие (рис. 2):

$$N_x = N\sin\alpha,\ N_y = N\cos\alpha.$$

Запишем теперь 2-й закон Ньютона отдельно для каждой координаты. Вдоль вертикальной оси наш объем ν находится в состоянии равновесия, поэтому суммарная сила, действующая на него в вертикальном направлении, должна быть равна нулю:

$$N\cos\alpha - mg = 0. \tag{1}$$

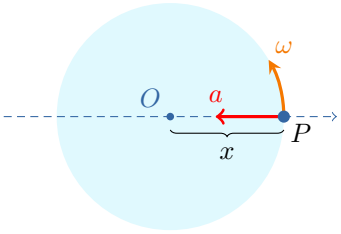


РИС. 3 Центростремительное ускорение для точки P (вид на сосуд сверху)

Единственная сила, действующая на объем ν вдоль горизонтального направления, — это проекция реакции опоры N_x. Так как данный объем совершает круговое движение с радиусом вращения x, то он испытывает центростремительное ускорение a, которое направлено горизонтально к оси вращения (рис. 3). Величина этого ускорения вычисляется по формуле (квадрат угловой скорости на радиус вращения):

$$a = \omega^2 x. \tag{2}$$

Таким образом, вдоль оси Ox второй закон Ньютона записывается в следующем виде:

$$-N\sin\alpha = -ma = -m\omega^2 x, \tag{3}$$

обе части взяты с отрицательным знаком, потому что и сила N_x, и ускорение a направлены против положительного направления оси Ox.

Запишем формулы (1) и (3) в виде системы двух уравнений:

$$\begin{aligned} Ox &: N\sin\alpha = m\omega^2 x, \\ Oy &: N\cos\alpha = mg. \end{aligned} \tag{4}$$

Поделим первое уравнение системы на второе:

$$\operatorname{tg}\alpha = \frac{\omega^2 x}{g}.$$

Теперь учтем тот факт, что согласно геометрическому смыслу производной функции $y(x)$ величина производной в точке x_0 равна тангенсу угла наклона касательной к соответствующей кривой $y(x)$ в заданной точке x_0. То есть мы можем заменить $\operatorname{tg}\alpha$ на y', что и даст нам искомое дифференциальное уравнение:

$$y' = \frac{\omega^2 x}{g}. \tag{5}$$

Если дополнительно обозначить высоту жидкости в точке $x = 0$ (на оси вращения) через y_0, то можно поставить для полученного дифференциального уравнения задачу Коши:

$$\begin{aligned} y' &= \frac{\omega^2 x}{g}, \\ y(0) &= y_0. \end{aligned} \tag{6}$$

Модель • Рассмотрим теперь процесс решения поставленной начальной задачи с помощью библиотеки SymPy. Сначала мы построим общее решение дифференциального уравнения (5), затем, используя начальное условие $y(0) = y_0$, решим начальную задачу (6). Следующим шагом определим параметр y_0 из условия постоянства объема вращаемой жидкости, рассмотрев при этом два случая — подкритический, когда поверхность вращаемой жидкости не касается дна цилиндра, и надкритический, когда такое касание происходит.

1 Построение модели начинаем с подключения библиотеки SymPy и функции display, определенной в модуле IPython.display.

```
from sympy import *
from IPython.display import display
```

2 С помощью команды symbols создаем символы x, g, ω, y_0 и C_1, которые нам потребуются для задания и решения дифференциального уравнения. Аргументом этой команды является строка, содержащая вводимые символы и определяющая, как эти символы будут выглядеть при печати. Возвращает эта команда кортеж с объектами SymPy, которые нужно запомнить в соответствующих переменных для последующего использования. Так как параметр g, представляющий собой ускорение свободного падения, является всегда строго положительным, то при создании соответствующего символа укажем этот факт, используя опциональный аргумент positive[2]. Используем команду display, чтобы посмотреть, как выглядят определенные нами символы. Например, видно, что SymPy корректно отображает греческие буквы, а цифры после символов преобразует в нижние индексы этих символов[3].

```
x, omega, y0, C1 = symbols("x omega y0 C1")
g = symbols("g", positive = True)
display(x, g, omega, y0, C1)
```

[2] Информация о положительности того или иного символа может быть использована библиотекой SymPy для упрощения формул, в которые входит этот символ, например, при извлечении квадратных корней.

[3] Для краткости вывод команды display в данном случае мы поместили в одну строку, на самом деле эта команда выводит каждое из переденных ей выражений в отдельной строке.

$$x\ g\ \omega\ y_0\ C_1$$

3 Наше дифференциальное уравнение

$$y' = \frac{\omega^2 x}{g}$$

является простейшим, это значит, что оно решается непосредственным интегрированием его правой части. Следовательно, для решения этого уравнения с помощью библиотеки SymPy нам достаточно определить только его правую часть, а не все уравнение целиком[4]. Создадим переменную ode_rhs, в которую и запишем правую часть решаемого уравнения.

[4] Как это мы будем делать в дальнейшем при решении более сложных дифференциальных уравнений.

```
ode_rhs = omega ** 2 * x / g
display(ode_rhs)
```

$$\frac{\omega^2 x}{g}$$

4 Общее решение простейшего дифференциального уравнения $y' = f(x)$ — это неопределенный интеграл от правой части $f(x)$ этого уравнения. В библиотеке SymPy неопределенные интегралы вычисляются с помощью команды integrate(f, x), где f — подынтегральное выражение, x — переменная интегрирования. Константа интегрирования C_1 при этом автоматически командой integrate к найденному интегралу не прибавляется, это надо делать вручную[5], поэтому мы просто к результату работы функции integrate прибавим символ C1. Найденное *общее решение* дифференциального уравнения сохраним в переменной dsol.

[5] Если это необходимо.

```
dsol = integrate(ode_rhs, x) + C1
display(dsol)
```

$$C_1 + \frac{\omega^2 x^2}{2g}$$

5 Стандартный способ определения константы интегрирования C_1 в общем решении дифференциального уравнения из начального условия заключается в подстановке в это решение величин из начального условия, что приводит к алгебраическому уравнению для переменной C_1. Решив это уравнение и подставив

найденное решение вместо C_1 обратно в формулу общего решения, мы найдем решение соответствующей начальной задачи для заданного дифференциального уравнения. В нашем случае начальное условие имеет вид $y(0) = y_0$, поэтому мы сначала подставляем в общее решение `dsol` вместо `x` значение `0`, это делается с помощью метода `subs(f, g)`, где `f` — выражение, которое мы хотим заменить на выражение `g`. Полученное в результате подстановки выражение приравниваем к величине `y0`, используя команду `Eq(lhs, rhs)`, которая создает равенство вида `lhs = rhs`[6]. Созданное уравнение запоминаем в переменной `eq1`.

```
eq1 = Eq(dsol.subs(x, 0), y0)
display(eq1)
```

$$C_1 = y_0$$

6 Следующим шагом решаем построенное уравнение с помощью команды `solveset(eq, x)`, где первый аргумент `eq` — это само уравнение, второй аргумент `x` — переменная, относительно которой данное уравнение должно решаться[7]. Найденное решение сохраним в переменной `sol1`.

```
sol1 = solveset(eq1, C1)
display(sol1)
```

$$\{y_0\}$$

7 Как мы видим, команда `solveset` выдает решения уравнения в виде множества, чтобы выбрать какое-то одно решение, нужно преобразовать это множество в кортеж (tuple) или список (list) и выбрать соответствующий элемент, указав его индекс. В нашем случае множество содержит всего одно решение, поэтому для его извлечения используем нулевой индекс. Запоминаем найденное значение в переменной `C2`.

```
C2 = tuple(sol1)[0]
display(C2)
```

$$y_0$$

8 Последним действием подставляем найденное значение `C2` вместо символа `C1` в общее решение `dsol`, что и дает нам искомое решение начальной задачи. Сохраним это решение в переменной `dsol_y0`.

```
dsol_y0 = dsol.subs(C1, C2)
display(dsol_y0)
```

6 `Python` не позволяет переопределить команду присваивания `=`, поэтому для создания уравнений приходится использовать специальную функцию `Eq`.

7 В рассматриваемом случае уравнение тривиально, но ровно такие же действия нужно будет выполнять и в более сложных случаях.

$$y_0 + \frac{\omega^2 x^2}{2g}$$

9 Как видно из последней формулы, найденная зависимость высоты y поверхности жидкости от расстояния до оси вращения x оказывается квадратичной, т. е. интегральные кривые (графики решений дифференциального уравнения) должны быть параболами[8]. Убедимся в этом, построив графики решения. Библиотека SymPy имеет несколько простых функций для построения графиков[9], простейшей из которых является функция plot. Построим с помощью этой функции график найденного нами решения начальной задачи при следующих значениях входящих в это решение констант:

$$y0 = 0,\ \omega = \pi/2,\ g = 9.8.$$

Чтобы сделать в некоторой формуле сразу несколько подстановок, можно создать из них словарь, ключами которого будут заменяемые символы или выражения, и подать этот словарь на вход методу subs.

```
ds = dsol_y0.subs({y0: 0, omega: pi/2, g: 9.8})
p1 = plot(ds)
```

Результат выполнения[10] этого фрагмента кода показан на рис. 4.

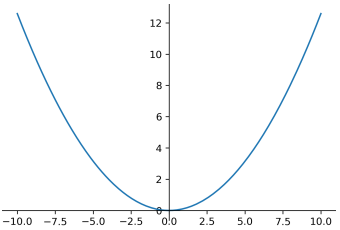


РИС. 4 График решения начальной задачи

10 Построим теперь графики семейства решений начальной задачи для различных значений угловой скорости $\omega \in [0, 2\pi]$ с шагом $\pi/10$. Для формирования списка значений переменной omega будем использовать функцию arange из библиотеки NumPy. Для построения нескольких кривых на одном графике нужно сначала создать пустой график и запомнить его в некоторой переменной, в нашем случае это будет переменная p2. Для каждой кривой следует создавать свой график (переменная p) и затем добавлять его к общему графику p2 с помощью метода extend. По умолчанию любой создаваемый график в среде Jupyter сразу же визуализируется, чтобы избежать этого, нужно в команде plot использовать опцию show = False. Построенный график можно визуализировать с помощью вызова метода show.

```
from numpy import arange
p2 = plot(show = False)
for om in arange(0, 2 * pi, pi / 10):
```

8 Таким образом, искомая поверхность будет парабалоидом вращения.

9 Являющихся, по сути, надстройкой над существенно более мощной графической библиотекой Matplotlib.

10 Переменная pi является предопределенной константой в SymPy.

```
    ds = dsol_y0.subs({y0: 0, omega: om, g: 9.8})
    p = plot(ds, (x, -1, 1), show = False)
    p2.extend(p)
p2.show()
```

Результат выполнения этого кода приведен на рис. 5.

11 При построении графиков решения выше мы *произвольно* задавали значение параметра `y0` равным нулю. Более естественный подход к определению этого параметра заключается в вычислении объема вращающейся жидкости и учете очевидного условия, что этот объем является постоянной величиной, не зависящей от скорости вращения ω. Пусть объем жидкости равен V_0. Чтобы выразить величину y_0 через V_0, нам надо вычислить объем вращающейся жидкости, приравнять его к V_0 и решить полученное уравнение относительно y_0. Заметим, что интересующий нас объем получается вращением участка кривой $y(x)$ на отрезке $[0, R]$ вокруг оси Oy, где R — радиус основания цилиндра (рис. 6). Следовательно, для его вычисления можно воспользоваться двойным интегралом в полярных координатах[11]:

$$V = \int_0^{2\pi} \int_0^R f(r)\, r dr d\varphi.$$

Роль радиуса r в нашем случае играет переменная x, а функция $f(r)$ представлена решением начальной задачи $y(x)$, следовательно,

$$V = \int_0^{2\pi} \int_0^R y(x)\, x dx d\varphi.$$

Данный двойной интеграл можно упростить с учетом того, что подынтегральное выражение не зависит от полярного угла φ:

$$V = 2\pi \int_0^R y(x)\, x dx. \qquad (7)$$

Введем два дополнительных символа для радиуса цилиндра и объема содержащейся в нем жидкости (оба параметра являются строго положительными величинами), вычислим *определенный* интеграл (7) и запомним результат в переменной `vol`. Определенные интегралы в библиотеке `SymPy` вычисляются с помощью той

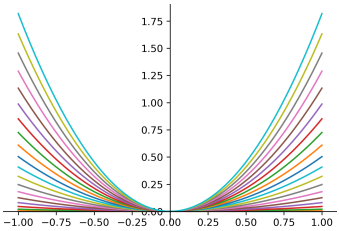


РИС. 5 График семейства решений начальной задачи для разных значений угловой скорости ω

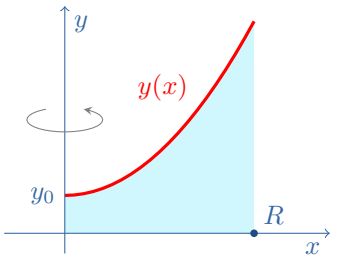


РИС. 6 Вычисление объема вращающейся жидкости

11 Наша поверхность является поверхностью вращения, поэтому функция $f(r)$ не зависит от полярного угла φ.

же функции `integrate`, но вторым ее аргументом для этого нужно указать кортеж из трех элементов: переменная интегрирования, нижний и верхний пределы интегрирования.

```
V0, R = symbols("V0 R", positive = True)
vol = 2 * pi * integrate(dsol_y0 * x, (x, 0, R))
display(vol)
```

$$2\pi\left(\frac{R^4\omega^2}{8g}+\frac{R^2y_0}{2}\right)$$

12 Приравниваем найденный объем к `V0` и получаем уравнение относительно переменной `y0`.

```
eq2 = Eq(vol, V0)
display(eq2)
```

$$2\pi\left(\frac{R^4\omega^2}{8g}+\frac{R^2y_0}{2}\right)=V_0$$

13 Решаем полученное уравнение относительно `y0`.

```
sol2 = solveset(eq2, y0)
display(sol2)
```

$$\left\{-\frac{\frac{R^4\omega^2}{g}-\frac{4V_0}{\pi}}{4R^2}\right\}$$

14 Уравнение `eq2` линейное, поэтому решение у него единственное, для его извлечения используем нулевой индекс. Заодно упростим полученную формулу, используя команду `simplify`.

```
y1 = simplify(tuple(sol2)[0])
display(y1)
```

$$-\frac{R^2\omega^2}{4g}+\frac{V_0}{\pi R^2}$$

15 Подставляем найденное выражение `y1` вместо символа `y0` в решение `dsol_y0` начальной задачи. Получаем еще одно выражение для решения этой же начальной задачи, но теперь выраженное через заданный объем жидкости `V0`.

```
dsol_V0 = dsol_y0.subs(y0, y1)
display(dsol_V0)
```

$$-\frac{R^2\omega^2}{4g}+\frac{\omega^2x^2}{2g}+\frac{V_0}{\pi R^2}$$

16 Построим еще раз семейство итегральных кривых нашей задачи для разных значений скорости вращения ω для случая $V_0 = 2$ (рис. 7).

```
p3 = plot(show = False)
for om in arange(0, 2 * pi, pi / 10):
    ds = dsol_V0.subs({V0:2, R:1, omega:om, g:9.8})
    p = plot(ds, (x, -1, 1), show = False)
    p3.extend(p)
p3.show()
```

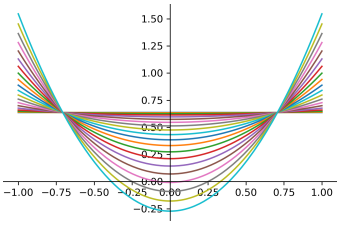


РИС. 7 График семейства решений начальной задачи для разных значений угловой скорости ω при фиксированном объеме V_0

17 Видно, что поведение интегральных кривых теперь стало более естественным — нижняя точка параболоида постепенно опускается при увеличении скорости вращения. Однако при достижении дна цилиндра ($y = 0$) это поведение становится нереалистичным — поверхность жидкости в некоторой области оказывается ниже дна цилиндра, а интегрирование по этой области дает отрицательный объем. То есть наши выкладки справедливы только при условии, что кривая $y(x)$ располагается строго над осью Ox. Найдем критическую скорость ω_0, при которой нижняя точка поверхности жидкости (при $x = 0$) касается дна цилиндра. Для этого подставим в решение $x = 0$, приравняем его к нулю и решим полученное уравнение относительно скорости вращения ω.

```
eq3 = Eq(dsol_V0.subs(x, 0), 0)
sol3 = solveset(eq3, omega)
display(sol3)
```

$$\left\{-\frac{2\sqrt{V_0}\sqrt{g}}{\sqrt{\pi}R^2},\frac{2\sqrt{V_0}\sqrt{g}}{\sqrt{\pi}R^2}\right\}$$

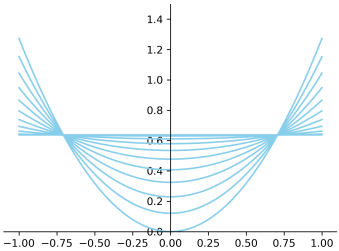


РИС. 8 График семейства решений начальной задачи в подкритическом случае

18 Команда `solve` в данном случае выдает два решения, отличающихся только знаком (т. е. направлением вращения), выберем из них положительное (с индексом 1) и сохраним его в переменной `omega0`. Построим графики интегральных кривых для случая $\omega < \omega_0$ (рис. 8). Этот случай назовем *подкритическим*. Для удобства заранее создадим словарь `par` для выполнения дальнейших подстановок. Опции `xlim` и `ylim` команды `plot` определяют пределы изменения переменных по горизонтальной и вертикальной осям графика. Кривые рисуем синим цветом, используя опцию `line_color` команды `plot`.

```
omega0 = tuple(sol3)[1]
display(omega0)
p4 = plot(ylim = (0, 1.5), show = False)
par = {V0: 2, omega: 0, R: 1, g: 9.8}
om0 = omega0.subs(par)
for om in arange(0, 1.1 * om0, 0.1 * om0):
    par[omega] = om
    ds = dsol_V0.subs(par)
    p = plot(ds, (x, -1, 1), line_color = "skyblue",
             show = False)
    p4.extend(p)
p4.show()
```

$$\frac{2\sqrt{V_0}\sqrt{g}}{\sqrt{\pi}R^2}$$

19 Рассмотрим теперь по аналогичной схеме *надкритический* случай, когда скорость вращения превышает критическую, что приводит к образованию в центре дна цилиндра области, не покрытой жидкостью. Первым шагом найдем радиус x_0 этой области. Для этого приравняем решение `dsol_y0` к нулю и решим полученное уравнение относительно x.

```
eq4 = Eq(dsol_y0, 0)
sol4 = solveset(eq4, x)
display(sol4)
```

$$\left\{-\frac{\sqrt{2}\sqrt{g}\sqrt{-y_0}}{\omega}, \frac{\sqrt{2}\sqrt{g}\sqrt{-y_0}}{\omega}\right\}$$

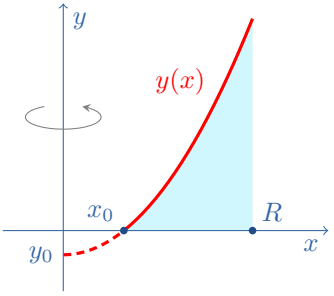


РИС. 9 Вычисление объема вращающейся жидкости в надкритическом случае

20 Из двух найденных решений выберем решение со знаком плюс. И еще раз вычислим объем, занимаемый жидкостью. Единственное отличие от предыдущего случая состоит в том, что интегрирование теперь выполняется от x_0 до R, т. е. часть ниже оси Ox мы не интегрируем (см. рис. 9):

$$V = 2\pi \int_{x_0}^{R} y(x)\, x dx.$$

```
x0 = tuple(sol4)[0]
display(x0)
vol1 = 2 * pi * integrate(dsol_y0 * x, (x, x0, R))
display(vol1)
```

$$\frac{\sqrt{2}\sqrt{g}\sqrt{-y_0}}{\omega}$$
$$2\pi\left(\frac{R^4\omega^2}{8g}+\frac{R^2y_0}{2}+\frac{gy_0^2}{2\omega^2}\right)$$

21 Приравниваем вычисленное выражение к V_0 и решаем полученное уравнение относительно y_0.

```
eq5 = Eq(vol1, V0)
sol5 = solveset(eq5, y0)
display(sol5)
```

$$\left\{-\frac{R^2\omega^2}{2g}-\frac{\sqrt{V_0}\omega}{\sqrt{\pi}\sqrt{g}},-\frac{R^2\omega^2}{2g}+\frac{\sqrt{V_0}\omega}{\sqrt{\pi}\sqrt{g}}\right\}$$

22 Получили два возможных значения параметра y_0. Проверим, какое из них оказывается равным нулю при подстановке вместо ω критического значения ω_0.

```
for s in tuple(sol5):
    display(s.subs(omega, omega0))
```

$$0$$
$$-\frac{4V_0}{\pi R^2}$$

23 Выбираем решение с индексом 0 и подставляем его в решение начальной задачи `dsol_y0` вместо символа `y0`, что дает нам уравнение поверхности для надкритического случая.

```
y2 = tuple(sol5)[0]
dsol_sup = dsol_y0.subs(y0, y2)
display(dsol_sup)
```

$$-\frac{R^2\omega^2}{2g}+\frac{\sqrt{V_0}\omega}{\sqrt{\pi}\sqrt{g}}+\frac{\omega^2x^2}{2g}$$

24 По аналогичной схеме строим семейство кривых (красного цвета) найденного решения для скорости $\omega\in[1.2\omega_0,3\omega_0]$ с шагом $0.2\omega_0$ (рис. 10).

```
p5 = plot(ylim = (0, 1.5), show = False)
for om in arange(1.2 * om0, 3 * om0, 0.2 * om0):
    par[omega] = om
    ds = dsol_sup.subs(par)
    p = plot(ds, (x, -1, 1), line_color = "red",
             show = False)
    p5.extend(p)
p5.show()
```

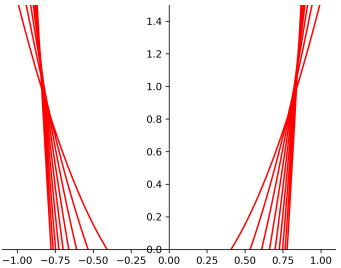


РИС. 10 График семейства решений начальной задачи в надкритическом случае

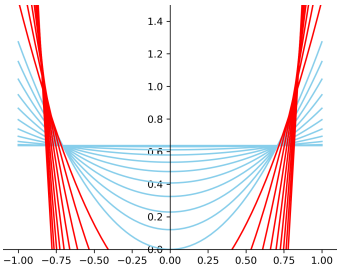


РИС. 11 График семейства решений начальной задачи в подкритическом (синие кривые) и надкритическом (красные кривые) случаях

ТАБЛ. 1 Стандартные математические функции в SymPy

Функция	SymPy
$\sqrt{x}$	`sqrt(x)`
e^x	`exp(x)`
$\ln x$	`log(x)`
$\sin x$	`sin(x)`
$\cos x$	`cos(x)`
$\operatorname{tg} x$	`tan(x)`
$\operatorname{ctg} x$	`cot(x)`
$\arcsin x$	`asin(x)`
$\arccos x$	`acos(x)`
$\operatorname{arctg} x$	`atan(x)`
$\operatorname{arcctg} x$	`acot(x)`

25 Наконец, объединим графики двух рассмотренных нами случаев: подкритического (переменная `p4`) и надкритического (переменная `p5`).

```
p6 = plot(ylim = (0, 1.5), show = False)
p6.extend(p4)
p6.extend(p5)
p6.show()
```

Итоговый график, включающий оба случая, показан на рис. 11.

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Решите с помощью SymPy следующие квадратные уравнения:

1) $x^2 - 4x + 3 = 0$;
2) $x^2 - 5x + 1 = 0$;
3) $64x^2 + 16x + 1 = 0$;
4) $2x^2 - 3x + 2 = 0$.

2 Решите с помощью SymPy следующие кубические уравнения, выведите на экран найденные *положительные* корни:

1) $x^3 - 3x^2 + 3x - 1 = 0$;
2) $x^3 - 7x + 6 = 0$;
3) $x^3 - 2x^2 - 7x - 4 = 0$;
4) $x^3 + 7x^2 + 14x + 8 = 0$.

3 Дифференцирование выражений в SymPy выполняется командой `diff(f, x)`, где `f` — выражение, `x` — переменная дифференцирования. Вычислите с помощью этой функции производные по x следующих выражений (см. табл. 1):

1) $5x^4 + x^2 - 4$;
2) $ax^2 + bx + c$;
3) $e^x \cdot \sqrt{4x + 2}$;
4) $\arccos(a^2 - x^2)$.

4 Вычислите с помощью команды `integrate` следующие *неопределенные* интегралы:

1) $\int (x^2 - 2x + 3)dx$;
2) $\int y \sin(y^2)dy$;
3) $\int se^s ds$;
4) $\int e^{2t} \cos t dt$.

5 Вычислите с помощью команды `integrate` следующие *определенные* интегралы[12]:

1) $\int_{-1}^{1} (x^3 - 1)dx$;
2) $\int_{a}^{b} \frac{y-1}{y+1} dy$;
3) $\int_{0}^{2\pi} \sin 2\varphi \, d\varphi$;
4) $\int_{1}^{\infty} e^{-3t} dt$.

[12] Библиотека SymPy частично поддерживает работу с символом бесконечности ∞, который определен в ней под именем `oo`.

6 Найдите с помощью `SymPy` общее решение простейшего дифференциального уравнения и решите соответствующую начальную задачу:

1) $y' = \dfrac{1}{x}$, $y(1) = -1$;
2) $y' = 2\cos^2 x$, $y(\pi) = 0$;
3) $y' = \dfrac{1 + 2x^2}{\sqrt{x}}$, $y(1) = -\dfrac{1}{5}$;
4) $y' = x \ln x$, $y(\sqrt{e}) = 1$.

Числовые дроби вида n/m с целыми n и m в `SymPy` задаются командой `Rational(n, m)`.

7 Постройте семейство кривых решения начальной задачи `dsol_y0` для значений параметра y_0 из интервала $[0, 2]$ с шагом 0.1 при фиксированной скорости вращения $\omega = 1$.

8 Постройте семейство кривых решения начальной задачи для разных значений ускорения свободного падения g, список этих значений сформируйте из величин ускорения свободного падения на поверхности различных космических тел Солнечной системы (Солнце, планеты, Луна).

9 Для подкритического случая найдите положение *неподвижной* точки x_1, в которой пересекаются все интегральные кривые рассматриваемого дифференциального уравнения. Может ли эта точка находиться вне цилиндра?

10 Напишите функцию для построения графика решения начальной задачи, автоматически определяющую, к какому случаю относится решение при заданных значениях параметров V_0, R, g и ω.

11 Постройте модель вращения жидкости в сосуде, имеющем форму параболоида вращения $y = kx^2$, $k > 0$.

12 Рассмотрите модель вращения жидкости в круговом цилиндре высоты H, который закрыт сверху крышкой, в предположении, что поверхность жидкости касается верхнего основания цилиндра при угловой скорости $\omega_1 < \omega_0$ (т. е. в подкритическом случае, см. рис. 12).

13 Рассмотрите случай вращения сосуда в форме тонкого прямоугольного параллелепипеда, толщина которого δ настолько мала, что искривлением поверхности вдоль соответствующей стороны можно пренебречь (рис. 13). Вращение сосуда выполняется вокруг оси, проходящей через его центр перпендикулярно основанию. Объем жидкости во вращающемся сосуде в данном случае вычисляется по

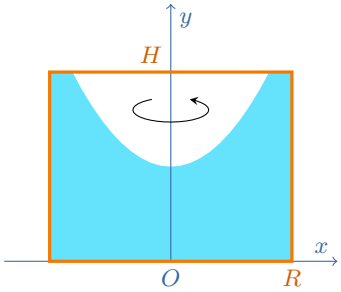


РИС. 12 Вращение жидкости в закрытом сверху цилиндре

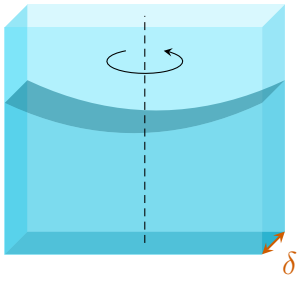


РИС. 13 Вращение тонкого параллелепипеда

более простой формуле:

$$V = 2\delta \int_0^R y(x)dx, \tag{8}$$

где R — половина длинной стороны основания параллелепипеда. Так как объем V постоянен и δ является константой, то постоянным должен быть интеграл в формуле (8). Найдите величину y_0 в предположении, что объем жидкости равен V_0. Рассмотрите по аналогии с построенной нами моделью подкритический и надкритический случаи.

14 Исследуйте модель вращения жидкости в тонком прямоугольном параллелепипеде (см. предыдущее упражнение) для случая, когда ось вращения смещена относительно центра параллелепипеда (см. рис. 14).

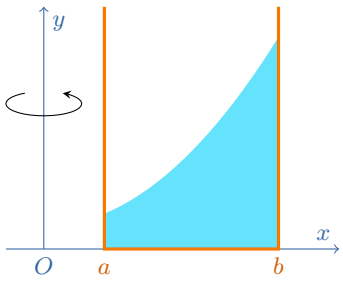


РИС. 14 Модель со смещенной осью вращения

ГЛАВА 2

Водяные часы

Эванджелиста Торричелли

Задача • В 1641 году Торричелли сформулировал закон вытекания жидкости из отверстий в стенке открытого сосуда и вывел формулу для определения скорости этого вытекания, называемую сейчас формулой Торричелли. Этот закон утверждает, что скорость v истечения жидкости через тонкое отверстие, находящееся на глубине h от поверхности жидкости, такая же, как и у тела, *свободно* падающего с высоты h (рис. 1). Если начальная скорость падения равна нулю, то за время t тело пролетит расстояние $h = gt^2/2$ и приобретет скорость $v = gt$ (согласно формулам равноускоренного движения). Исключив из этих двух равенств время t, найдем, что скорость v тела связана с расстоянием h (высотой падения) следующей формулой:

$$v = \sqrt{2gh}. \qquad (1)$$

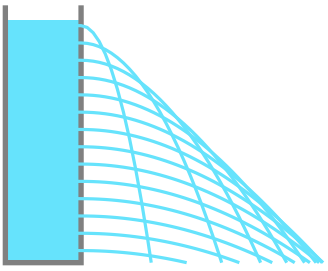


РИС. 1 Вытекание жидкости из сосуда с разных высот согласно формуле Торричелли (1)

Пусть у нас имеется сосуд, заполненный водой до высоты H, в нижней части которого сделано отверстие площади σ (рис. 2). Требуется определить закон, по которому изменяется со временем высота $h(t)$ воды в данном сосуде. Предполагается, что нам задана форма сосуда в виде зависимости площади S сечения сосуда от высоты h этого сечения.

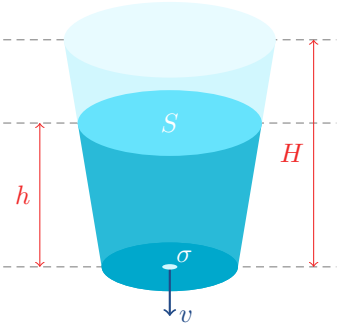


РИС. 2 Модель вытекания воды из сосуда

Рассмотрим, как изменяется объем воды в сосуде за время от t до $t+\Delta t$. С одной стороны, за указанный промежуток времени Δt из сосуда вытечет объем воды, равный $\Delta V = v\sigma\Delta t$. Предполагается, что скорость истечения воды за данный промежуток времени практически не меняется, и согласно формуле Торричелли эта скорость равна $v = \sqrt{2gh}$. Следовательно, объем

воды снизится на величину

$$\Delta V = \sigma\sqrt{2gh}\Delta t. \tag{2}$$

С другой стороны, уровень воды за этот же промежуток времени снизится с h до $h+\Delta h$ (т. е. $\Delta h < 0$). Это значит, что ее объем уменьшится на величину

$$\Delta V = -S(h)\Delta h, \tag{3}$$

по аналогичным соображениям мы предполагаем, что площадь S за рассматриваемый промежуток времени также практически не меняется. Приравнивая величины (2) и (3) и выполняя предельный переход при $\Delta t \to 0$, приходим к дифференциальному уравнению следующего вида:

$$S(h)dh = -\sigma\sqrt{2gh}dt \;\Rightarrow\; S(h)\frac{dh}{dt} = -\sigma\sqrt{2gh}. \tag{4}$$

Решить в общем виде это уравнение мы, однако, пока не можем, т. к. процесс этого решения существенным образом зависит от вида функции $S(h)$. Если же нам эта функция задана, то, подставляя ее в уравнение (4), мы получим уже полностью определенное дифференциальное уравнение, которое можно решать тем или иным способом.

Решение $h(t)$ уравнения (4), если мы сможем его найти, позволит нам построить специальную шкалу, с помощью которой можно по текущему уровню воды определять время, прошедшее с момента открытия сливного отверстия. По сути, это превращает наш сосуд в широко применявшиеся (вплоть до XVII века) водяные часы, которые в Древней Греции были известны под названием *клепсидры* (рис. 3).

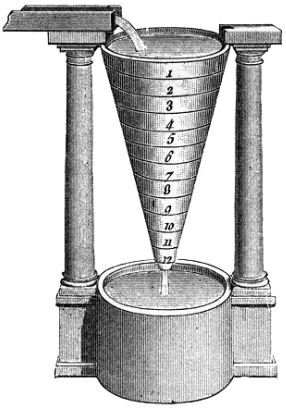


РИС. 3 Водяные часы

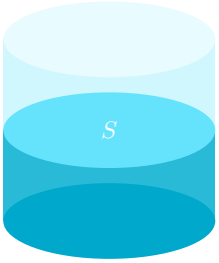


РИС. 4 Цилиндрический сосуд

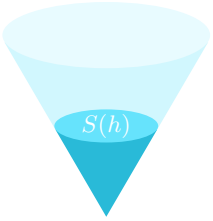


РИС. 5 Сосуд в форме конуса

Модель • Процесс решения построенного дифференциального уравнения (4) с помощью `SymPy` рассмотрим на примере сосудов двух видов — в форме кругового цилиндра (рис. 4), когда площадь любого горизонтального сечения является постоянной, и в форме конуса (рис. 5), когда радиус поперечного сечения линейно убывает с высотой (т. е. площадь убывает с высотой квадратично).

1 Построение модели стандартно начинаем с подключения библиотек.

```
from sympy import *
from IPython.display import display
from numpy import arange
```

2 Определяем символы σ, g, t и S[1], входящие в дифференциальное уравнение (4), каждый из которых является неотрицательной величиной.

```
sigma, g, t, S = symbols("sigma g t S",
                         positive = True)
display(sigma, g, t, S)
```

$$\sigma \; g \; t \; S$$

3 Для создания и решения дифференциальных уравнений с помощью SymPy нам необходимо уметь определять объекты, представляющие собой неизвестные функции. Такие объекты в SymPy создаются с помощью команды Function[2].

```
h = Function("h")(t)
display(h)
```

$$h(t)$$

4 Кроме того, нам потребуется производная нашей неизвестной функции $h(t)$, которую мы создадим с помощью команды Derivative[3]. По принятому в SymPy неформальному соглашению[4] производная заданной функции обозначается символом этой функции и символом подчеркивания.

```
h_ = Derivative(h, t)
display(h_)
```

$$\frac{d}{dt}h(t)$$

5 Теперь мы можем определить наше дифференциальное уравнение в общей форме.

```
ode = Eq(S * h_, -sigma * sqrt(2 * g) * sqrt(h))
display(ode)
```

$$S\frac{d}{dt}h(t) = -\sqrt{2}\sqrt{g}\sigma\sqrt{h(t)}$$

[1] Вместо S мы будем в дальнейшем подставлять разные выражения, соответствующие той или иной форме сосуда.

[2] Если команды Symbol и symbols вводят так называемые свободные переменные, то команда Function определяет зависимую переменную, т. е. функцию.

[3] Команда diff *вычисляет* производную заданного выражения, а команда Derivative *создает* формулу для производной.

[4] Которого вы, в принципе, можете и не придерживаться.

6 Пусть наш сосуд имеет форму цилиндра с постоянной площадью поперечного сечения (рис. 4):

$$S(h) = S = \text{const}\,.$$

В этом случае введенный нами символ `S` становится просто параметром модели, а дифференциальное уравнение `ode` оказывается полностью определенным. В библиотеке `SymPy` общее решение дифференциального уравнения `eq` относительно неизвестной функции `x` ищется с помощью команды `dsolve(eq, x)`. Найдем общее решение нашего дифференциального уравнения и сохраним результат в переменной `dsol_cyl`.

```
1 dsol_cyl = dsolve(ode, h)
2 display(dsol_cyl)
```

$$h(t) = \frac{C_1^2 - 2\sqrt{2}C_1\sqrt{g}\sigma t + 2g\sigma^2 t^2}{4S^2}$$

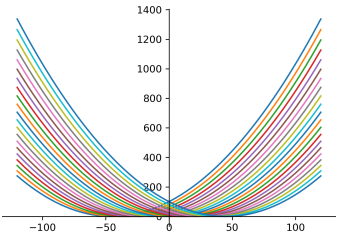


РИС. 6 Семейство интегральных кривых дифференциального уравнения 4 для случая $S = \text{const}$

7 Построенное общее решение зависит от константы интегрирования C_1, значение которой можно определить из начального условия. Чтобы получить доступ к этой константе, нужно определить соответствующий символ `C1`. Для примера построим для различных значений константы C_1 семейство интегральных кривых нашего дифференциального уравнения при некоторых значениях параметров модели (рис. 6). Заметим, что решение, выдаваемое командой `dsolve`, является *равенством*, т. е. объектом типа `Eq`: чтобы извлечь его правую часть (которая, собственно, и представляет искомое решение дифференциального уравнения), надо использовать атрибут `rhs`[5] объекта `Eq` (строка 6).

```
1 C1 = symbols("C1")
2 par = {g: 9.8, sigma: 0.1, S: 1.0, C1: 0.0}
3 p1 = plot(show = False)
4 for c in arange(-20, 21, 2):
5     par[C1] = c
6     ds = dsol_cyl.rhs.subs(par)
7     p = plot(ds, (t, -120, 120), show = False)
8     p1.extend(p)
9 p1.show()
```

8 Хотя построенные нами графики являются математически корректными, с точки зрения рассматриваемой физической модели они выглядят нереалистично — высота со временем должна убывать (жидкость *вытекает* из сосуда), а наши графики показывают, что

[5] От первых букв словосочетания right had side. Соответственно, левая часть равенства выделяется с помощью атрибута `lhs`.

после полного вытекания жидкости из сосуда ее высота начинает *расти*. Кроме того, остается неясным физический смысл константы C_1, намного более наглядным будет использование вместо этой константы интегрирования какой-нибудь осмысленной величины, например начальной высоты жидкости[6]:

$$h(0) = H. \tag{5}$$

Итак, определим символ `H` (положительная величина) и подставим начальное условие (5) в общее решение `dsol_cyl`, что даст нам уравнение относительно переменной `C1`.

```
H = symbols("H", positive = True)
eq1 = dsol_cyl.subs({h: H, t: 0})
display(eq1)
```

$$H = \frac{C_1^2}{4S^2}$$

9 Решим это уравнение, решение сохраним в переменной `sol1`.

```
sol1 = solveset(eq1, C1)
display(sol1)
```

$$\left\{-2\sqrt{H}S, 2\sqrt{H}S\right\}$$

10 Выберем решение со знаком плюс, запомним его в переменной `C2`[7]. Подставим выбранное выражение в общее решение `dsol_cyl` вместо символа `C1`, результат подстановки (решение начальной задачи) сохраним в переменной `dsol_cyl_H`.

```
C2 = tuple(sol1)[1]
display(C2)
dsol_cyl_H = dsol_cyl.subs(C1, C2)
display(dsol_cyl_H)
```

$$2\sqrt{H}S$$

$$h(t) = \frac{-4\sqrt{2}\sqrt{H}S\sqrt{g}\sigma t + 4HS^2 + 2g\sigma^2 t^2}{4S^2}$$

11 Вычислим время полного вытекания жидкости из сосуда, заменив в найденном решении высоту `h` на ноль и решив полученное алгебраическое уравнение относительно t.

[6] Это выражение, очевидно, служит начальным условием для рассматриваемого дифференциального уравнения.

[7] При выборе решения из множества с помощью преобразования этого множества в кортеж рекомендуется проверять, правильное ли решение вы выбрали. Порядок решений в кортеже может отличаться от того порядка, в котором решения отображаются командой `display`.

```
eq2 = dsol_cyl_H.subs(h, 0)
sol2 = solveset(eq2, t)
display(sol2)
```

$$\left\{\frac{\sqrt{2}\sqrt{H}S}{\sqrt{g}\sigma}\right\}$$

12 Решение единственное, извлечем его и запомним в переменной `T1`. Теперь мы можем построить графики решения начальной задачи в интервале от нуля до `T1` (предварительно выполнив в `T1` подстановку числовых параметров).

```
T1 = tuple(sol2)[0]
par = {g: 9.8, sigma: 0.1, S: 1.0, H: 0.0}
p2 = plot(show = False)
for ht in arange(1, 21, 1):
    par[H] = ht
    ds = dsol_cyl_H.rhs.subs(par)
    t1 = T1.subs(par)
    p = plot(ds, (t, 0, t1), show = False)
    p2.extend(p)
p2.show()
```

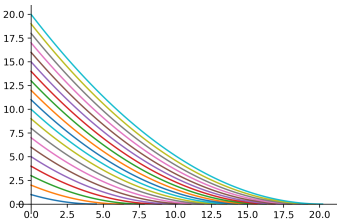


РИС. 7 Семейство кривых решения начальной задачи для случая $S = \text{const}$ и разных значений начальной высоты $h(0) = H$

Результат выполнения этого кода приведен на рис. 7.

13 Теперь перейдем ко второй модели, в которой рассмотрим вытекание жидкости из сосуда в форме прямого кругового конуса (рис. 5). В этом случае площадь поперечного сечения является функцией от высоты h. Предположим, что радиус R поперечного сечения равен высоте h этого сечения. Вычислим площадь сечения (площадь круга радиуса R) на данной высоте, результат запомним в переменной `s`.

```
R = h
s = pi * R ** 2
display(s)
```

$$\pi h^2(t)$$

14 Подставим в исходное дифференциальное уравнение вместо символа `S` найденную площадь `s`.

```
ode_cone = ode.subs(S, s)
display(ode_cone)
```

$$\pi h^2(t)\frac{d}{dt}h(t) = -\sqrt{2}\sqrt{g}\sigma\sqrt{h(t)}$$

15 Решим построенное уравнение. В текущей версии SymPy данное уравнение командой `dsolve(ode_cone, h)`, к сожалению, не решается. В таких случаях рекомендуется использовать так называемые подсказки (hints) для команды `dsolve`. В нашем случае мы видим, что уравнение `ode_cone` является уравнением с разделяющимися переменными, поэтому попробуем решить его, используя подсказку `separable`, которая передается в команду `dsolve` в виде опции `hint`.

```
dsol_cone = dsolve(ode_cone, h, hint = "separable")
display(dsol_cone)
```

```
[Eq(h(t),2**(3/5)*(C1-5*sqrt(2)*sqrt(g)*sigma*t)**(2/5)/(2*pi**(2
/5))), Eq(h(t),(C1-sqrt(2)*sqrt(g)*sigma*t)**(2/5)*(-2**(3/5)*5**
(9/10)-2**(3/5)*5**(2/5)-2**(1/10)*5**(9/10)*I*sqrt(sqrt(5)+5)+2*
*(1/10)*5**(2/5)*I*sqrt(sqrt(5)+5))/(8*pi**(2/5))), Eq(h(t),(C1-s
qrt(2)*sqrt(g)*sigma*t)**(2/5)*(-2**(3/5)*5**(9/10)-2**(3/5)*5**(
2/5)-2**(1/10)*5**(2/5)*I*sqrt(sqrt(5)+5)+2**(1/10)*5**(9/10)*I*s
qrt(sqrt(5)+5))/(8*pi**(2/5))), Eq(h(t),(C1-sqrt(2)*sqrt(g)*sigma
*t)**(2/5)*(-2**(3/5)*5**(2/5)+2**(3/5)*5**(9/10)+2**(1/10)*5**(2
/5)*I*sqrt(5-sqrt(5))+2**(1/10)*5**(9/10)*I*sqrt(5-sqrt(5)))/(8*p
i**(2/5))), Eq(h(t),(C1-sqrt(2)*sqrt(g)*sigma*t)**(2/5)*(-2**(3/5
)*5**(2/5)+2**(3/5)*5**(9/10)-2**(1/10)*5**(9/10)*I*sqrt(5-sqrt(5
))-2**(1/10)*5**(2/5)*I*sqrt(5-sqrt(5)))/(8*pi**(2/5)))]
```

16 Как видно, команда `dsolve` выдает *список* сразу из пяти решений[8], последние четыре из которых, впрочем, оказываются комплексными, т. к. содержат в себе мнимую единицу, которая в SymPy обозначается символом `I`[9]. Выделим первое решение (с индексом ноль) и запомним его в той же переменной `dsol_cone`.

```
dsol_cone = dsol_cone[0]
display(dsol_cone)
```

$$h(t) = \frac{2^{\frac{3}{5}}\left(C_1 - 5\sqrt{2}\sqrt{g}\sigma t\right)^{\frac{2}{5}}}{2\pi^{\frac{2}{5}}}$$

17 Дальше действуем по схеме предыдущей модели. Подставляем в найденное решение начальное условие $h(0) = H$ и решаем полученное алгебраическое уравнение относительно константы `C1`, результат запоминаем в `sol3`.

```
eq3 = dsol_cone.subs({h: H, t: 0})
sol3 = solveset(eq3, C1)
display(sol3)
```

[8] Пять объектов `Eq` с левой частью `h(t)`.

[9] Эти комплексные решения являются следствием извлечения корней пятой степени и того факта, что SymPy все свои вычисления по умолчанию выполняет в комплексных числах.

$$\left\{\frac{2\pi H^{\frac{5}{2}}}{5}\right\}$$

18 Извлекаем из `sol3` единственное решение и подставляем его в общее решение `dsol_cone` вместо символа `C1`, полученное частное решение сохраняем в переменной `dsol_cone_H`.

```
C3 = tuple(sol3)[0]
dsol_cone_H = dsol_cone.subs(C1, C3)
display(dsol_cone_H)
```

$$h(t) = \frac{2^{\frac{3}{5}}\left(2\pi H^{\frac{5}{2}} - 5\sqrt{2}\sqrt{g}\sigma t\right)^{\frac{2}{5}}}{2\pi^{\frac{2}{5}}}$$

19 Составляем и решаем уравнение для времени T_2 полного вытекания жидкости из сосуда.

```
eq4 = dsol_cone_H.subs(h, 0)
sol4 = solveset(eq4, t)
display(sol4)
```

$$\left\{\frac{\sqrt{2}\pi H^{\frac{5}{2}}}{5\sqrt{g}\sigma}\right\}$$

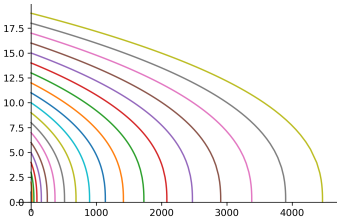


РИС. 8 Семейство кривых решения начальной задачи для сосуда в форме конуса и разных значений начальной высоты жидкости $h(0) = H$

20 Строим графики построенного решения начальной задачи для разных значений начальной высоты жидкости H. Каждый отдельный график строим в диапазоне времени от $t = 0$ ($h = H$) до $t = T_2$ ($h = 0$).

```
T2 = tuple(sol4)[0]
par = {g: 9.8, sigma: 0.1, H: 0.0}
p3 = plot(show = False)
for ht in arange(1, 20, 1):
    par[H] = ht
    ds = dsol_cone_H.rhs.subs(par)
    t2 = T2.subs(par)
    p = plot(ds, (t, 0, t2), show = False)
    p3.extend(p)
p3.show()
```

Результат построения приведен на рис. 5.

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Найдите с помощью SymPy общее решение заданного дифференциального уравнения с разделяющимися переменными:

1) $y' = y^2$;
2) $xy' \ln x = y$;
3) $xy + (x+1)y' = 0$;
4) $y' + 1 = xy^2 + x - y^2$.

2 Решите с помощью SymPy следующие начальные задачи для дифференциальных уравнений с разделяющимися переменными:

1) $y' = 2x^2y^2$, $y(1) = 2$;
2) $y' \operatorname{ctg} x + y = 2$, $y(\pi/3) = 0$;
3) $xy' - 2y = 0$, $y(1) = 1$;
4) $y' = 1 - ax$, $y(2) = 0$.

3 Библиотека SymPy всегда пытается найти решение заданного дифференциального уравнения в *явной* форме, что далеко не всегда возможно. Если не получается найти явное решение, то можно попытаться построить решение в *неявной* форме в виде некоторого соотношения, связывающего неизвестную функцию и независимую переменную, используя опцию `symplify = False` команды `dsolve`. Например, применение этой опции при решении дифференциального уравнения `ode_cone` приводит к следующему результату:

```
dsol_cone = dsolve(ode_cone, h, hint = "separable",
                   simplify = False)
display(dsol_cone)
```

$$\frac{2\pi h^{\frac{5}{2}}(t)}{5} = C_1 - \sqrt{2}\sqrt{g}\sigma t$$

4 Начальные задачи в SymPy можно решать сразу с помощью команды `dsolve`, используя ее опциональный аргумент `ics`. Начальные условия задачи[10] при этом оформляются в виде словаря, ключами которого служат левые части этих условий, а значениями — их правые части. Например, для начального условия $h(0) = H$ нашей задачи словарь `ic` будет состоять из единственного элемента с ключом $h(0)$ и значением H. Пример кода:

```
ic = {h.subs(t,0): H}
dsol_cone_H = dsolve(ode_cone, h, ics = ic,
                     hint = "separable",
                     simplify = False)
display(dsol_cone_H)
```

[10] Одно — для уравнений первого порядка, и несколько — для уравнений высших порядков.

$$\frac{2\pi h^{\frac{5}{2}}(t)}{5} = \frac{2\pi H^{\frac{5}{2}}}{5} - \sqrt{2}\sqrt{g}\sigma t$$

5 Решите по описанной в предыдущем упражнении схеме начальные задачи из упражнения 2.

6 Для построения графика найденного неявного решения (см. предыдущее упражнение) с помощью команды `plot`[11] надо предварительно преобразовать это решение в явную форму, что не всегда возможно. В нашем случае более простым вариантом является преобразование решения к виду $t(h)$[12] и построение соответствующего графика, горизонтальная ось которого будет соответствовать высоте, вертикальная — времени. Так как SymPy строит графики выражений, зависящих от символов, а не от функций, то предварительно в нашем неявном решении надо заменить функцию $h(t)$ на символ h (первые две строки кода), после этого выражаем время через высоту (третья строка) и строим график решения для заданного набора параметров (пятая строка), указывая пределы изменения высоты от нуля до начального значения H[13]:

```
1 _h = symbols("h")
2 dsol_cone_h = dsol_cone_H.subs(h, _h)
3 _t = tuple(solveset(dsol_cone_h, t))[0]
4 par = {g: 9.8, sigma: 0.1, H: 10.0, h: hh}
5 p = plot(_t.subs(par), (_h, 0, par[H]))
```

Результат выполнения данного фрагмента кода приведен на рис. 9.

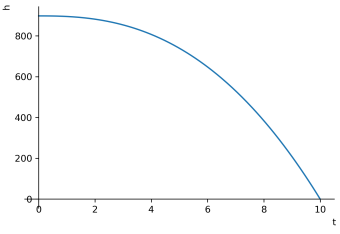


РИС. 9 График зависимости времени t от высоты h

7 В некоторых случаях имеет смысл решать дифференциальные уравнения с разделяющимися переменными в полностью ручном режиме[14]. Рассмотрим для примера решение дифференциального уравнения $xy' = xy + 2y$. Определяем символы для переменной x, функции y[15] и ее производной y' (и символ для константы C_1), после чего создаем дифференциальное уравнение.

```
1 x, y, y_, C1 = symbols("x y y' C1")
2 ode = Eq(x * y_ - x * y - 2 * y, 0)
3 display(ode)
```

$$xy' = xy + 2y$$

Разрешаем полученное дифференциальное уравнение относительно производной.

```
1 rhs = tuple(solveset(ode, y_))[0]
2 display(Eq(y_, rhs))
```

$$y' = \frac{xy + 2y}{x}$$

11 Библиотека SymPy имеет специальную команду `plot_implicit` для построения графиков неявных функций, мы рассмотрим пример работы этой команды в одной из следующих глав.

12 Время выражаем через высоту.

13 При таком способе построения графика нам не надо искать время полного вытекания, как это мы делали при построении графиков выше.

14 Без использования команды `dsolve`.

15 При описываемом подходе используются только символы, функции не нужны.

Раскладываем на множители правую часть, используя команду `separatevars`. Эта команда принимает на вход два обязательных параметра — выражение и список переменных, по которым должно выполняться разделение. Опциональный параметр `dict = True` указывает, что результат разделения должен быть представлен словарем, ключами в котором являются разделяемые переменные, а все множители, которые не содержат указанные переменные, объединяются в элемент словаря с ключом `'coeff'`.

```
dic = separatevars(rhs, [x, y], dict = True)
print(dic)
```

```
{x: (x + 2)/x, y: y, 'coeff': 1}
```

Разделяем переменные.

```
p = dic['coeff'] * dic[x]
q = dic[y]
display(Eq(Integral(1 / q, y), Integral(p, x)))
```

$$\int \frac{1}{y}\,dy = \int \frac{x+2}{x}\,dx$$

Вычисляем интегралы и формируем из них общее решение дифференциального уравнения в неявном виде[16].

```
Q = integrate(1 / q, y)
P = integrate(p, x)
dsol = Eq(Q, P + C1)
display(dsol)
```

$$\log(y) = C_1 + x + 2\log(x)$$

Решите по описанной схеме дифференциальные уравнения из упражнения 1.

8 Обобщите построенные в настоящей главе модели на случай сосудов, являющихся телами вращения, радиус сечений которых зависит от высоты следующим образом:

$$r(h) = r_0 h^k,$$

где h — высота сечения, r_0 и k — параметры. Например, для цилиндра $k = 0$, а для конуса $k = 1$. Исследуйте поведение интегральных кривых для других значений k: $k = 1/2$ (параболоид вращения), $k = 2$ и т. д. При каком значении k скорость изменения высоты будет постоянной[17]? Что будет происходить с моделью при отрицательных значениях k?

9 Для форм из предыдущего упражнения постройте зависимость времени T полного вытекания жидкости из сосуда от ее начальной высоты H.

[16] При необходимости можно попытаться найти решение и в явной форме, используя команду `solveset(dsol, y)`.

[17] То есть для измерения времени с помощью сосуда такой формы можно будет использовать обычную линейку с равномерной шкалой.

10 Рассмотрите перевернутые варианты форм из упражнения 8 с заданной высотой H. Сравните время полного вытекания жидкости из двух вариантов каждой рассмотренной формы (прямой и перевернутой).

11 Постройте модель вытекания жидкости из сосудов в форме: 1) усеченного конуса с заданными высотой и радиусами оснований; 2) верхней и нижней полусфер заданного радиуса; 3) сферы заданного радиуса.

12 Постройте и исследуйте модель вытекания жидкости из сосудов, в которых горизонтальными сечениями являются прямоугольники, у которых длина одной стороны a фиксирована, а длина второй b меняется с высотой по степенному закону $b = b_0 h^k$.

РИС. 10 Формы с прямоугольными сечениями для упражнения 12 (слева направо: $k = 0.5$, $k = 1$, $k = 2$)

13 Сравните время полного вытекания *одинакового* объема жидкости из сосудов разных форм.

14 Интересной и нереалистичной особенностью построенной нами модели вытекания жидкости из сосуда в форме конуса является то, что скорость изменения высоты *увеличивается неограниченно* при уменьшении высоты:

$$\frac{dh}{dt} \to \infty \text{ при } h \to 0.$$

Это значит, что при достижении некоторой критической высоты h_c скорость изменения высоты превысит, например, скорость света. Найдите это значение h_c.

15 Исследуйте зависимость времени полного вытекания жидкости из сосуда заданной формы от ускорения свободного падения g. Где жидкость будет вытекать из одного и того же сосуда быстрее — на Земле или на Луне?

ГЛАВА 3

Элементарные химические реакции

Закон действующих масс • *Элементарной химической реакцией* называется реакция, протекающая в одну стадию (без образования промежуточных веществ) при непосредственном взаимодействии молекул исходных веществ друг с другом. В таких реакциях, как правило, образуется или разрывается не более одной-двух связей между атомами взаимодействующих молекул, например

$$\mathrm{H_2 + O \to H_2O}. \qquad (1)$$

Уравнения вида (1) называются *стехиометрическими*. В общем виде они записываются следующим образом:

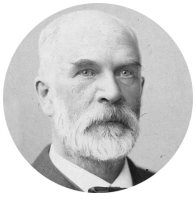
$$n_1\mathrm{A}_1 + n_2\mathrm{A}_2 + \dots \to m_1\mathrm{X}_1 + m_2\mathrm{X}_2 + \dots, \qquad (2)$$

где A_i — реагенты, X_i — продукты реакции, n_i и m_i — *стехиометрические коэффициенты*, которые показывают, сколько молекул каждого вида участвует в реакции с той или другой стороны.

Число реагирующих молекул называется *молекулярностью* реакции. Для элементарных реакций эта величина также называется их *порядком*[1]. Элементарные реакции обычно имеют первый или второй порядок, существенно реже — третий. Элементарных реакций более высоких порядков не бывает в силу исчезающе малой вероятности одновременного столкновения четырех и более молекул.

Закон действующих масс, открытый в 1864 году норвежскими учеными Гульдбергом и Вааге, утверж-

[1] В сложных реакциях порядок, вообще говоря, не равен молекулярности.



Като Максимилиан Гульдберг

Петер Вааге

дает, что скорость v протекания элементарной реакции (2), т. е. количество элементарных актов реакции в единицу времени, пропорциональна концентрациям реагирующих веществ в степенях, равных стехиометрическим коэффициентам:

$$v = k a_1^{n_1} a_2^{n_2} \dots, \tag{3}$$

где a_i — концентрация[2] молекул вида A_i, k — коэффициент пропорциональности (константа скорости реакции), специфичный для данной реакции и зависящий также от условий проведения реакции, например от температуры раствора.

[2] Количество молекул на единицу объема, измеряется в моль/л.

Скорость реакции определяет скорость, с которой изменяется концентрация каждого вещества, участвующего в этой реакции:

$$\frac{da_i}{dt} = \delta_i \cdot v, \tag{4}$$

где параметр δ_i показывает изменение числа молекул рассматриваемого вида в результате одного элементарного акта реакции. Например, для реакции вида

$$\mathrm{A} + 2\mathrm{B} \to 3\mathrm{C} \tag{5}$$

скорость реакции равна

$$v = kab^2, \tag{6}$$

где a и b — концентрации веществ A и B соответственно[3]. Тогда скорости изменения концентраций веществ A, B и C в данной реакции будут равны соответственно:

$$\frac{da}{dt} = -v, \quad \frac{db}{dt} = -2v, \quad \frac{dc}{dt} = 3v, \tag{7}$$

[3] Мы и дальше будем придерживаться такой системы обозначений: заглавная латинская буква обозначает вещество, соответствующая ей строчная буква — концентрацию этого вещества.

т. к. в процессе одного элементарного акта рассматриваемой реакции (5) число молекул A уменьшается на одну, поэтому $\delta_\mathrm{A} = -1$, число молекул B уменьшается на две, поэтому $\delta_\mathrm{B} = -2$, а число молекул C увеличивается на три, поэтому $\delta_\mathrm{C} = 3$.

Дифференциальные уравнения вида (7) называются *кинетическими*. Основная задача химической кинетики заключается в определении того, как для заданной системы химических реакций меняются со временем концентрации реагирующих веществ.

Задача • В реакциях второго порядка участвуют две молекулы — либо одинаковые, либо разные. Поэтому такие реакции называют *бимолекулярными*. В качестве модельной рассмотрим следующую элементарную химическую реакцию второго порядка:

$$\mathrm{A} + \mathrm{B} \to 2\mathrm{C}. \tag{8}$$

Пусть соответствующие начальные концентрации равны a_0, b_0 и c_0. Согласно закону действующих масс скорость реакции (8) равна

$$v = kab. \tag{9}$$

Тогда скорости изменения концентраций веществ A, B и C равны

$$\frac{da}{dt} = -kab, \quad \frac{db}{dt} = -kab, \quad \frac{dc}{dt} = 2kab. \tag{10}$$

Обозначим убыль концентрации a через x:

$$x(t) = a_0 - a(t) \Rightarrow a(t) = a_0 - x(t). \tag{11}$$

Тогда, согласно стехиометрическому уравнению реакции (8) убыль вещества B и прибыль вещества C равны x и $2x$ соответственно:

$$b(t) = b_0 - x(t), \quad c(t) = c_0 + 2x(t). \tag{12}$$

Подставляем выражения (11) и (12) в (10) и в начальное условие и приходим к одной и той же начальной задаче относительно $x(t)$:

$$\begin{aligned} -\frac{dx}{dt} &= -k(a_0 - x)(b_0 - x), \\ x(0) &= 0. \end{aligned} \tag{13}$$

Модель • Рассмотрим процесс построения описанной выше модели реакции (8) в библиотеке `SymPy`. Причем в этот раз мы начнем не с решения дифференциального уравнения, а с его вывода, чтобы потом можно было обобщить рассматриваемую модель на случай других химических реакций.

1 Подключаем необходимые библиотеки[4].

[4] См. первые две модели.

2 Вводим символы для переменной t, константы скорости реакции k, начальных концентраций a_0, b_0 и c_0. Определяем функцию $x(t)$. Функции $a(t)$, $b(t)$ и $c(t)$ как объекты `SymPy` нам не понадобятся, т. к. мы будем выражать их через $x(t)$, для чего будет достаточно обычных переменных. По этой же причине не нужно определять производную функции $x(t)$.

3 Определяем изменения δ_A, δ_B и δ_C числа молекул наших трех веществ в одной химической реакции.

```
da, db, dc = -1, -1, 2
print(da, db, dc)
```

```
-1 -1 2
```

4 Функция $x(t)$ обозначает убыль вещества a, т. е. $x(t) = a_0 - a(t)$, следовательно, $a(t) = a_0 - x(t)$.

```
a = a0 - x
display(a)
```

$$a_0 - x(t)$$

5 Теперь мы можем выразить функции $b(t)$ и $c(t)$ через убыль $x(t)$. Для этого заметим, что на одну молекулу вещества A, расходуемую в одной реакции, расходуется δ_B/δ_A молекул вещества B и δ_C/δ_A молекул вещества C. Если какое-то из этих отношений является положительным, то соответствующие молекулы расходуются, если отрицательным — то создаются.

```
b = b0 - db * x / da
c = c0 - dc * x / da
display(b, c)
```

$$b_0 - x(t)\ \ c_0 + 2x(t)$$

6 Вычисляем скорость реакции v.

```
v = k * a * b
display(v)
```

$$k\,(a_0 - x(t))\,(b_0 - x(t))$$

7 Записываем кинетическое уравнение $\dot{a} = kv$ для концентрации $a(t)$, производную $\dot{a}$ при этом *вычисляем* с помощью команды `diff`. Библиотека `SymPy` автоматически подставит вместо переменных `a` и `v` вычисленные ранее их выражения через функцию `x`, и в результате мы получаем искомое дифференциальное уравнение для неизвестной функции $x(t)$.

```
ode = Eq(diff(a, t), da * v)
display(ode)
```

$$-\frac{d}{dt}x(t) = -k\left(a_0 - x(t)\right)\left(b_0 - x(t)\right)$$

8 Создаем нулевое начальное условие $x(0) = 0$ и решаем соответствующую начальную задачу.

```
ic = {x.subs(t, 0): 0}
dsol = dsolve(ode, x, ics = ic)
display(dsol)
```

$$x(t) = \frac{-a_0 e^{b_0\left(kt + \frac{\log\left(\frac{a_0}{b_0}\right)}{a_0 - b_0}\right)} + b_0 e^{a_0\left(kt + \frac{\log\left(\frac{a_0}{b_0}\right)}{a_0 - b_0}\right)}}{e^{a_0\left(kt + \frac{\log\left(\frac{a_0}{b_0}\right)}{a_0 - b_0}\right)} - e^{b_0\left(kt + \frac{\log\left(\frac{a_0}{b_0}\right)}{a_0 - b_0}\right)}}$$

9 Подставим в найденное параметризованное решение конкретные значения константы скорости реакции и начальных концентраций.

```
par = {k: 1, a0: 1, b0: 2, c0: 0}
ds = dsol.subs(par)
display(ds)
```

$$x(t) = \frac{-4e^{2t} + 4e^{t}}{-4e^{2t} + 2e^{t}}$$

10 Построим график найденного решения (рис. 1) на интервале $t \in [0, T]$ для значения $T = 5$.

```
T = 5
p1 = plot(ds.rhs, (t, 0, T), show = False)
p1.show()
```

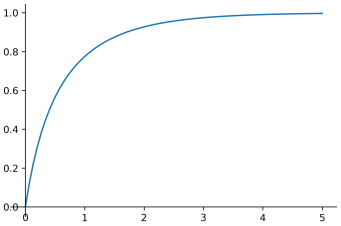


РИС. 1 Зависимость $x(t)$ для случая $k = 1$, $a_0 = 1$, $b_0 = 2$ и $c_0 = 0$

11 Последним действием построим графики искомых зависимостей $a(t)$, $b(t)$ и $c(t)$. Все три кривые строятся по одной и той же схеме (поэтому мы будем их строить в цикле): заменяем в соответствующем выражении символ `x` на правую часть найденного решения `dsol`; заменяем там же параметры на их числовые значения, используя определенный выше словарь `par`; строим на интервале $t \in [0, T]$ заданным цветом график полученной зависимости; добавляем построенную кривую к общему графику `p2`.

```
p2 = plot(show = False)
L = [(a, "red"), (b, "blue"), (c, "green")]
for r, cl in L:
    rs = r.subs(x, dsol.rhs).subs(par)
    p = plot(rs, (t, 0, T),
             line_color = cl, show = False)
    p2.extend(p)
p2.show()
```

Результат выполнения приведенного кода показан на рис. 2.

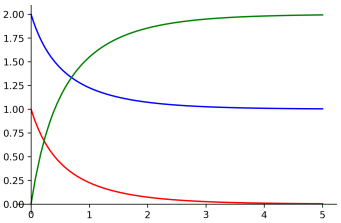


РИС. 2 Зависимость концентраций $a(t)$, $b(t)$ и $c(t)$ для случая $k = 1$, $a_0 = 1$, $b_0 = 2$ и $c_0 = 0$

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Если проанализировать решение `dsol` дифференциального уравнения `ode`, то оказывается, что оно было получено библиотекой `SymPy` в неявном предположении, что $a_0 \neq b_0$. При $a_0 = b_0$ решением уравнения будет рациональная функция, а не комбинация экспонент. Если все-таки условие $a_0 = b_0$[5] требуется учесть, то это можно сделать двумя способами. Во-первых, можно явно заменить b_0 на a_0 при *создании* дифференциального уравнения. Во-вторых, можно сначала выполнить подстановку числовых параметров, а затем уже решать конкретное непараметризованное уравнение.

[5] Это условие является в некотором смысле вырожденным, т. к. добиться на практике *точного* совпадения двух концентраций невозможно.

2 Постройте и проанализируйте модели для элементарных химических реакций первого порядка:

1) $A \to B$ — реакция *изомеризации*;
2) $A \to B + C$ — реакция *диссоциации*.

3 Постройте модели для следующих элементарных химических реакций второго порядка:

1) $2A \to B$;
2) $2A \to A + B$;
3) $A + B \to C + D$;
4) $A + B \to C$.

4 Постройте и решите кинетическое уравнение для реакции второго порядка с *катализатором*:

$$A + C \to B + C. \tag{14}$$

Как следует из (14), катализатор C в реакции не расходуется, т. е. $c(t) \equiv c_0$, но влияет на ее скорость (рис. 3).

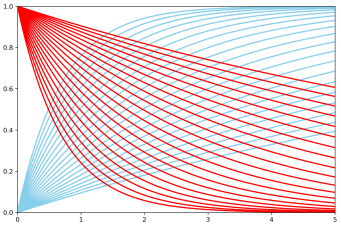


РИС. 3 Графики зависимостей $a(t)$ (красная кривая) и $b(t)$ (голубая кривая) для реакции (14), показывающие влияние концентрации катализатора c_0 на скорость реакции. Какие кривые соответствуют большим значениям c_0?

5 Исследуйте модель *автокаталитической* реакции следующего вида

$$A + C \to C + C, \tag{15}$$

в которой катализатор реакции C является также ее продуктом.

6 Рассмотрим обратимую реакцию первого порядка:

$$\mathrm{A} \leftrightarrow \mathrm{B}, \tag{16}$$

состоящую из двух отдельных реакций $\mathrm{A} \to \mathrm{B}$ и $\mathrm{B} \to \mathrm{A}$, протекающих со скоростями

$$v_1 = k_1 a \quad \text{и} \quad v_2 = k_2 b$$

соответственно. Чтобы найти скорости изменения концентраций a и b, можно воспользоваться *принципом независимости*: если в системе имеется несколько простых реакций, то каждая из них протекает по таким же кинетическим законам и с теми же скоростями, как и в отсутствие других реакций. Следовательно,

$$\frac{da}{dt} = -k_1 a + k_2 b, \quad \frac{db}{dt} = k_1 a - k_2 b. \tag{17}$$

Так как в данном случае уравнение материального баланса тривиально[6]:

$$a + b = a_0 + b_0, \tag{18}$$

то систему также можно свести к одному дифференциальному уравнению для $x(t) = a_0 - a(t)$. Выполните такое построение в SymPy и решите полученное дифференциальное уравнение.

[6] Суммарное количество молекул A и B остается постоянным.

7 Постройте и проанализируйте модель обратимой каталитической реакции:

$$\mathrm{A} + \mathrm{C} \to \mathrm{B} + \mathrm{C}, \ \mathrm{B} \to \mathrm{A}, \tag{19}$$

в которой константы скоростей равны k_1 и k_2 соответственно.

8 Во многих приложениях важным оказывается не столько решение кинетического уравнения, сколько нахождение его предельного значения:

$$x_\infty = \lim_{t \to \infty} x(t).$$

Величина x_∞ называется точкой покоя соответствующего дифференцильного уравнения, с ее помощью можно найти предельные распределения уже концентраций всех веществ, участвующих в реакции (в системе реакций). В библиотеке SymPy пределы вычисляются с помощью команды `limit(f, x, a)`, где `f` — выражение, предел которого мы ищем, `x` — переменная, `a` — значение, к которому стремится переменная `x`. Причем вместо `a` можно использовать символы бесконечности `oo` и `-oo`. Найдите с помощью этой команды предельные значения концентраций веществ для реакций из упражнений 6 и 7.

9 Вычислите с помощью `SymPy` следующие пределы:

1) $\lim\limits_{x\to 4} \dfrac{x-4}{\sqrt{x}-2}$;
2) $\lim\limits_{x\to 1} \dfrac{\cos x}{\sin^2 \pi x}$;
3) $\lim\limits_{x\to \infty} x(e^{\frac{1}{x}}-1)$;
4) $\lim\limits_{x\to 0} \dfrac{1-\cos ax}{x^2}$.

10 Выразите из найденного решения $x(t)$ кинетического дифференциального уравнения (для любой из рассмотренных вами моделей) переменную t через функцию x, например, с помощью команды `solveset` (предварительно преобразовав символ функции в символ переменной, см. упражнение 6 в предыдущей главе). Используя найденную зависимость, ответьте на следующий вопрос: через какое время убыль заданного вещества составит половину от его исходного количества? Такое время является в некотором смысле характеристикой скорости протекания реакции.

11 Постройте дифференциальное уравнение для элементарной химической реакции третьего порядка

$$\mathrm{A} + \mathrm{B} + \mathrm{C} \to \mathrm{D} \tag{20}$$

с начальным условием $a(0) = a_0$, $b(0) = b_0$ и $c(0) = c_0$. Решите поставленную начальную задачу для различных значений начальных концентраций.

ГЛАВА 4

Задача о четырех жуках

Кривые погони • *Кривыми погони*, или *кривыми преследования*, называются траектории движения объектов, выполняющих преследование подвижных или неподвижных целей согласно некоторому алгоритму. Если алгоритм описывается простым правилом, например «держать направление движения строго на цель», то кривая погони часто может быть построена с помощью решения некоторого дифференциального уравнения.

Классической задачей преследования является задача о четырех жуках[1]. В начальный момент времени жуки располагаются в вершинах квадрата (рис. 1). Они начинают одновременно двигаться с одинаковой постоянной скоростью. Каждый жук ползет в направлении своего соседа против часовой стрелки. Требуется определить, по каким траекториям будут двигаться жуки и какова будет длина этих траекторий от момента старта до момента встречи всех жуков в центре квадрата, если скорость жуков постоянна и равна v. Естественно, размерами жуков пренебрегаем, полагая их безразмерными точками.

Задача • Рассмотрим сразу общую постановку задачи, в которой n жуков расположены в начальный момент времени в вершинах правильного n-угольника (рис. 2). Введем систему координат: начало координат поместим в центр n-угольника, ось Ox направим на первого жука, ось Oy — перпендикулярно оси Ox (тогда точка старта второго жука должна иметь положительную y-координату). Для краткости рассуждений предположим, что расстояние от начала координат до каждого из жуков равно 1.

Эдуард Люка

[1] В оригинальном варианте этой задачи, поставленной в 1877 году французским математиком Эдуардом Люка, рассматривались три собаки, расположенные в вершинах правильного треугольника.

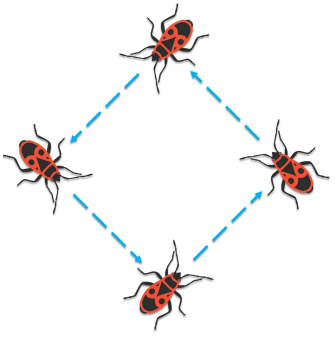


РИС. 1 Задача о четырех жуках

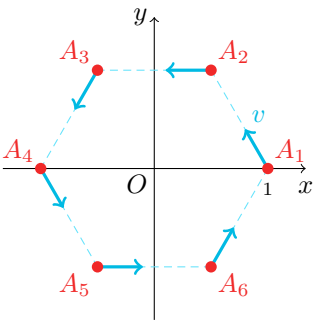


РИС. 2 Начальное состояние в задаче об n жуках (для случая $n = 6$)

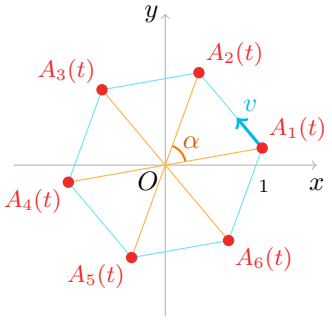


РИС. 3 Взаимное расположение жуков в произвольный момент времени t ($n = 6$)

Очевидно, что в силу симметрии задачи в каждый момент времени жуки располагаются в вершинах некоторого правильного n-угольника с центром в начале координат. Следовательно, если координаты первого жука (точка A_1) в некоторый момент времени равны (x, y), то координаты (x_1, y_1) его соседа против часовой стрелки (точка A_2) в тот же момент времени можно получить поворотом вектора OA_1 на угол $\alpha = 2\pi/n$ против часовой стрелки (см. рис. 3).

Воспользовавшись стандартными формулами линейной алгебры для поворотов в двумерном пространстве, выразим координаты точки A_2 через координаты (x, y) точки A_1 и угол α:

$$\begin{aligned} x_1 &= x\cos\alpha - y\sin\alpha, \\ y_1 &= x\sin\alpha + y\cos\alpha. \end{aligned} \tag{1}$$

По условию задачи скорость жука, находящегося в точке (x, y), направлена на его соседа, т. е. сонаправлена вектору

$$(x_1 - x, y_1 - y).$$

Значит, компоненты скорости должны быть равны

$$\begin{aligned} \frac{dx}{dt} &= k(x_1 - x), \\ \frac{dy}{dt} &= k(y_1 - y), \end{aligned} \tag{2}$$

где k — коэффициент, зависящий от скорости v и от расстояния между жуками[2].

Поделим в системе (2) второе уравнение на первое, рассматривая производные как отношения дифференциалов:

$$\frac{dy}{dx} = \frac{y_1 - y}{x_1 - x} = \frac{x\sin\alpha + y\cos\alpha - y}{x\cos\alpha - y\sin\alpha - x}. \tag{3}$$

Получили дифференциальное уравнение, описывающее кривую преследования каждого из n жуков. Траектория движения конкретного жука определяется с помощью начального условия: она должна проходить через его точку старта. Например, для первого жука начальное условие имеет следующий вид (рис. 2):

$$y(1) = 0. \tag{4}$$

[2] Коэффициент k равен отношению скорости v и длины вектора A_1A_2. Нетрудно убедиться, что в этом случае вектор $(\dot{x}, \dot{y})$ сонаправлен вектору A_1A_2 и имеет длину v, что полностью соответствует условиям задачи.

Модель • Рассмотрим процесс решения с помощью библиотеки `SymPy` дифференциального уравнения (3) с начальным условием (4). По ряду причин, которые будут указаны ниже, решать данное уравнение мы будем с помощью перехода к полярной системе координат, т. е. с помощью одновременной замены и неизвестной функции, и независимой переменной.

1 Подключаем необходимые библиотеки.

2 Определяем символы `x` и `alpha`, а также функцию `y(x)` и ее производную `y_`.

3 Вычисляем по формулам (1) координаты (x_1, y_1) точки A_2, куда должна быть направлена скорость первого жука, находящегося в точке A_1 с координатами (x, y).

```
x1 = x * cos(alpha) - y * sin(alpha)
y1 = x * sin(alpha) + y * cos(alpha)
display(x1, y1)
```

$$x\cos(\alpha) - y(x)\sin(\alpha)$$
$$x\sin(\alpha) + y(x)\cos(\alpha)$$

4 Составляем дифференциальное уравнение (3), используя вычисленные координаты x_1 и y_1.

```
ode = Eq(y_, (y1 - y) / (x1 - x))
display(ode)
```

$$\frac{d}{dx}y(x) = \frac{x\sin(\alpha) + y(x)\cos(\alpha) - y(x)}{x\cos(\alpha) - x - y(x)\sin(\alpha)}$$

5 Построенное уравнение относится к типу однородных дифференциальных уравнений первого порядка[3]. Проверим это с помощью команды `classify_ode`.

```
classify_ode(ode, y)
```

```
1st_homogeneous_coeff_best
1st_homogeneous_coeff_subs_indep_div_dep
1st_homogeneous_coeff_subs_dep_div_indep
1st_power_series
lie_group
1st_homogeneous_coeff_subs_indep_div_dep_Integral
1st_homogeneous_coeff_subs_dep_div_indep_Integral
```

[3] Уравнение вида $y' = f(x, y)$ называется однородным, если правая часть зависит только от отношения y/x: $f(x, y) = F(y/x)$.

Как видно, SymPy правильно определяет тип уравнения (первые три строчки вывода соответствуют разным вариациям именно однородных дифференциальных уравнений). Но, к сожалению, решить это уравнение с помощью команды dsolve не получается. Причиной этого является наличие параметра α в уравнении. Само уравнение после соответствующей замены сводится к уравнению с разделяющимися переменными, в процессе разделения переменных возникает рациональная функция с квадратичным знаменателем, коэффициенты которого зависят от α. Анализ дискриминанта этого знаменателя (знак которого определяет способ интегрирования всей дроби) оказывается в результате очень сложной задачей для случая произвольного α.

6 Так как исходная задача обладает очевидной поворотной симметрией (поворот всей конфигурации на любой угол, кратный α, приводит к той же самой конфигурации, просто с другой нумерацией жуков), то следует ожидать, что в полярной системе координат эта задача должна решаться более естественно, чем в декартовой системе координат. Выполним такую замену переменных в SymPy. Для этого сначала введем новый символ для независимой переменной φ и определим новую неизвестную функцию $r(\varphi)$.

```
phi = symbols("phi")
r = Function("r")(phi)
display(r)
```

$$r(\varphi)$$

7 Замене подлежат три величины: x, y и производная y'. Первые две величины выражаются через φ и r с помощью стандартных формул, связывающих декартовы и полярные координаты заданной точки:

$$\begin{aligned} x &= r\cos\varphi, \\ y &= r\sin\varphi. \end{aligned} \tag{5}$$

Для замены производной y' воспользуемся следующим приемом[4]:

$$y' = \frac{dy}{dx} = \frac{dy}{d\varphi}\cdot\frac{d\varphi}{dx} = \frac{\frac{dy}{d\varphi}}{\frac{dx}{d\varphi}}. \tag{6}$$

[4] Используя формулы для производной сложной функции и для производной обратной функции.

Создадим три соответствующие переменные для каждой из перечисленных формул с учетом того, что r — это функция угла φ.

```
zx = r * cos(phi)
zy = r * sin(phi)
zy_ = diff(zy, phi) / diff(zx, phi)
display(zx, zy, zy_)
```

$$r(\varphi)\cos(\varphi)$$
$$r(\varphi)\sin(\varphi)$$
$$\frac{r(\varphi)\cos(\varphi)+\sin(\varphi)\frac{d}{d\varphi}r(\varphi)}{-r(\varphi)\sin(\varphi)+\cos(\varphi)\frac{d}{d\varphi}r(\varphi)}$$

8 Выполняем замену, подставляя в уравнение `ode` выражения `zx`, `zy` и `zy_` вместо символов `x`, `y` и `y_` соответственно. Запомним новое дифференциальное уравнение в переменной `ode1`.

```
ode1 = ode.subs({y_: zy_, y: zy, x: zx})
display(ode1)
```

$$\frac{r(\varphi)\cos(\varphi)+\sin(\varphi)\frac{d}{d\varphi}r(\varphi)}{-r(\varphi)\sin(\varphi)+\cos(\varphi)\frac{d}{d\varphi}r(\varphi)} =$$
$$= \frac{r(\varphi)\sin(\alpha)\cos(\varphi)+r(\varphi)\sin(\varphi)\cos(\alpha)-r(\varphi)\sin(\varphi)}{-r(\varphi)\sin(\alpha)\sin(\varphi)+r(\varphi)\cos(\alpha)\cos(\varphi)-r(\varphi)\cos(\varphi)}$$

9 Визуально уравнение `ode1` выглядит намного хуже, чем исходное уравнение `ode`. Однако это уравнение в отличие от исходного легко решается[5], и решение имеет очень простой вид. Решаем уравнение с учетом начального условия $r(0)=1$: первый жук в начальный момент времени располагается на оси Ox, т. е. $\varphi=0$, на расстоянии $r=1$ от начала координат.

```
ic = {r.subs(phi, 0): 1}
dsol = dsolve(ode1, r, ics = ic)
display(dsol)
```

$$r(\varphi) = e^{\frac{\varphi(\cos(\alpha)-1)}{\sin(\alpha)}}$$

10 Построим график найденной траектории для случая $n=6$. Библиотека `SymPy` не поддерживает построение графиков в полярной системе координат, поэтому мы воспользуемся командой `plot_parametric` для построения графиков параметрически заданных функций. Первым аргументом этой команды указывается

[5] Несмотря на наличие в уравнении нелинейных функций $\sin\varphi$ и $\cos\varphi$, они в конце концов все сокращаются, и оказывается, что уравнение является *линейным* (см. замечание 3 к главе).

кортеж из двух выражений для x и y координат, выраженных через некоторый параметр (в нашем случае через параметр φ), вторым аргументом — кортеж, содержащий символ параметра и диапазон его изменения. Сохраняем в переменной `R` правую часть найденного решения, подставив туда предварительно угол $\alpha = 2\pi/n$. Далее выражаем x и y через φ по формулам (5). Устанавливаем предел `phi_max` для угла φ. Наконец, строим график, опция `size` задает размеры изображения[6].

```
n = 6
a = 2 * pi / n
R = dsol.rhs.subs(alpha, a)
X = R * cos(phi)
Y = R * sin(phi)
phi_max = 4 * pi
p1 = plot_parametric((X, Y), (phi, 0, phi_max),
                     xlim = (-1, 1), ylim = (-1, 1),
                     size=(6, 6), show = False)
p1.show()
```

Результат построения показан на рис. 4.

11 Построим теперь на одном графике траектории всех n жуков. Заметим, что траектория k-го жука получается поворотом траектории первого жука (которую мы нарисовали в предыдущем пункте) на угол $k\alpha$ против часовой стрелки. Создаем пустой график `p2`. Проходим в цикле по всем жукам. На k-й итерации этого цикла делаем следующие действия: выполняем поворот траектории `R` на угол $k\alpha$ с помощью замены φ на $\varphi - k\alpha$ (строка 4); получаем параметризованное описание траектории в декартовых координатах (строки 5 и 6); определяем пределы изменения угла φ для k-го жука — он стартует с угла $k\alpha$ (строка 7); строим траекторию и добавляем ее к общему графику (строки 8–10).

```
p2 = plot(xlim = (-1, 1), ylim = (-1, 1),
          size = (6, 6), show = False)
for k in range(n):
    R2 = R.subs(phi, phi - k * a)
    X = R2 * cos(phi)
    Y = R2 * sin(phi)
    phi1, phi2 = k * a, k * a + phi_max
    p = plot_parametric((X, Y), (phi, phi1, phi2),
                        show = False)
    p2.extend(p)
p2.show()
```

Окончательный результат показан на рис. 5.

[6] В данном случае существенно, чтобы масштаб по обеим осям был одинаковым, для этого мы используем по обеим осям одинаковые пределы и одинаковые размеры.

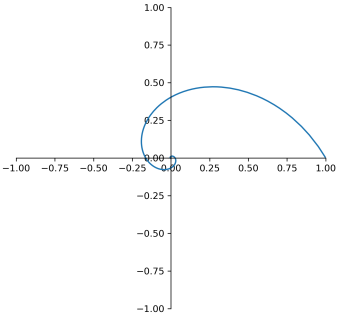


РИС. 4 Траектория первого жука для случая $n = 6$

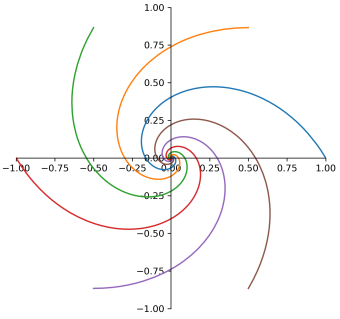


РИС. 5 Траектории движения всех жуков для случая $n = 6$

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Если воспользоваться известной формулой тангенса половинного аргумента

$$\operatorname{tg}\frac{\alpha}{2}=\frac{1-\cos\alpha}{\sin\alpha},$$

то найденное нами в пункте 9 решение дифференциального уравнения можно записать в более простом виде:

$$r(\varphi)=e^{-k\varphi},\ \text{где}\ k=\operatorname{tg}\frac{\alpha}{2}. \tag{7}$$

2 Найденная нами кривая погони называется логарифмической спиралью. Впервые такие спирали были описаны Декартом, искавшим кривую, обладающую свойством, подобным свойству окружности: касательная в каждой точке должна образовывать с радиус-вектором в этой точке один и тот же угол[7]. Декарт показал, что это условие равносильно тому, что полярные углы для точек кривой пропорциональны логарифмам радиус-векторов, решение (7) как раз и удовлетворяет этому свойству. Позже логарифмическая спираль исследовалась Якобом Бернулли, который называл её «удивительной спиралью» — *Spira mirabilis*. Якоб Бернулли хотел, чтобы на его могиле была выгравирована логарифмическая спираль, но вместо этого по ошибке туда поместили *архимедову* спираль (рис. 6).

3 Полученное нами дифференциальное уравнение в полярных координатах (см. пункт 8) можно упростить, разрешив его относительно производной $dr/d\varphi$ с помощью команды `solveset`:

```
rhs = solveset(ode1, diff(r, phi))
display(rhs)
```

$$\left\{\frac{(\cos(\alpha)-1)\,r(\varphi)}{\sin(\alpha)}\right\}\setminus\left\{\frac{r(\varphi)\sin(\varphi)}{\cos(\varphi)}\right\}$$

Получили одно решение (первая часть ответа), но с ограничением на область допустимых решений (вторая часть). Если пренебречь ограничением, то можно записать исходное уравнение в более простой форме:

```
ode2 = Eq(diff(r, phi), rhs.args[0].args[0])
display(ode2)
```

$$\frac{d}{d\varphi}r(\varphi)=\frac{(\cos(\alpha)-1)\,r(\varphi)}{\sin(\alpha)}$$

[7] Для окружности этот угол оказывается прямым.

Рене Декарт

Якоб Бернулли

РИС. 6 Архимедова спираль на надгробии Якоба Бернулли

Для извлечения части решения мы использовали стандартный атрибут `args`, в котором хранятся все части выражения (слагаемые, множители, элементы множества и т. д.). Первое обращение `args[0]` к этому атрибуту выделяет из `rhs` первую часть ответа (множество из одного элемента), второе аналогичное обращение выделяет из этого выделенного множества первый его элемент[8].

[8] Таким образом, подобный способ можно использовать и для выделения нужных решений из множества решений, выдаваемых командой `solveset`, вместо преобразования этого множества в кортеж или список.

4 Найдите траекторию k-го жука не поворотом траектории первого жука, как мы это делали выше, а напрямую через решение начальной задачи для дифференциального уравнения `ode1` с начальным условием:

$$r\big((k-1)\alpha\big) = 1, \text{ где } \alpha = 2\pi/n. \tag{8}$$

5 Логарифмическая спираль делает вокруг начала координат бесконечно много оборотов. Чтобы при построении траекторий не подбирать вручную подходящее значение параметра `phi_max`, можно его вычислять с помощью решения алгебраического уравнения $r(\varphi) = r_0$, где $r(\varphi)$ — решение `ode1` дифференциального уравнения, r_0 — минимальное расстояние от спирали до начала координат, например $r_0 = 0.01$. Протестируйте этот прием сначала на большом значении $r_0 = 0.5$.

6 Относительно простым частным случаем, который разрешим в декартовых координатах, является классический случай с $n = 4$ жуками. Подставьте в исходное дифференциальное уравнение `ode` (в декартовых координатах) $\alpha = \pi/2$ и решите полученное уравнение. Попробуйте построить график найденного решения в неявной форме с помощью команды `plot_implicit`. Сравните график в неявной форме с графиком в полярных координатах.

7 Дифференциальное уравнение для задачи с двумя жуками ($n = 2$) в полярных координатах не решается, но в этот раз успешно решается в декартовых. Проверьте оба этих утверждения. Какая кривая будет траекторией каждого из жуков в данном случае?

8 Работает ли наша модель для случая $n = 1$?

9 Проверьте, что следующие дифференциальные уравнения являются однородными (homogeneous), и решите их с помощью `SymPy`:

1) $x + yy' = 0$;
2) $2xyy' = y^2 - x^2$;
3) $x^2y' = y^2$;
4) $(y + 2x)y' = 2y - x$.

10 Решите дифференциальные уравнения из предыдущего упражнения с помощью перехода к полярной системе координат.

11 Постройте графики следующих кривых в полярной системе координат в указанных пределах угла φ:

1) $r(\varphi) = \dfrac{\varphi}{4}$, $\varphi \in [0, 8\pi]$;
2) $r(\varphi) = 1 + \cos\varphi$, $\varphi \in [0, 2\pi]$;
3) $r(\varphi) = 1 + 2\sin^2\dfrac{3\varphi}{2}$, $\varphi \in [0, 6\pi]$;
4) $r(\varphi) = \operatorname{tg}\dfrac{\varphi}{3}$, $\varphi \in [-3\pi, 3\pi]$.

12 Постройте: график заданной параметрической функции $\big(x(t), y(t)\big)$ в указанных пределах изменения параметра t для случая $a = 1$; семейство таких графиков для $a \in [0..2]$ с шагом 0.1.

1) $x = a\sin 2t$, $y = \cos 3t$, $t \in [0, 2\pi]$;
2) $x = a\sin t$, $y = (1 + a)\cos 3t$, $t \in [-2\pi, 0]$;
3) $x = a\sin 3t$, $y = \cos t + \cos 5t$, $t \in [-\pi, \pi]$;
4) $x = \cos t + a\sin 2t$, $y = a\sin 5t$, $t \in [0, \pi]$.

13 Хотя логарифмическая спираль в нашем случае и делает бесконечно много оборотов вокруг начала координат, ее длина является конечной. Найдите длину найденной траектории, используя формулу для длины кривой в полярной системе координат:

$$L = \int_0^\infty \sqrt{r^2 + r'^2}\,d\varphi. \tag{9}$$

Пределы интеграла соответствуют пределам изменения полярного угла, в нашем случае (для первого жука) этот угол изменяется от нуля до бесконечности. Для случая $n = 6$ вы должны получить $L = 2$.

14 Рассмотрим дискретный вариант задачи об n жуках. В каждый дискретный момент времени $t = 0, 1, 2, \ldots$ все жуки синхронно делают один шаг в направлении своих соседей против часовой стрелки, величина шага пропорциональна расстоянию до соответствующего соседа. Если коэффициент пропорциональности δ мал, то мы получим достаточно хорошее приближение к логарифмической спирали. Для демонстрации такой модели построим картинку, аналогичную той, которая использовалась на обложке журнала Scientific American в 1965 году (рис. 7). Так как модель дискретна, то для построения графиков мы будем использовать библиотеку `Matplotlib`. Подключаем библиотеку и импортируем из модуля `math` константу `pi` и функции `sin` и `cos`[9]. Задаем число жуков n. Вычисляем угол α. Создаем два списка с координатами жуков: первый

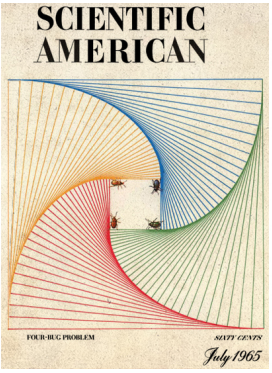


РИС. 7 Обложка журнала Scientific American за июль 1965 года

[9] Это стандартные объекты `Python`, потому что в данной модели мы не работаем с символьными вычислениями.

жук располагается в точке $(1, 0)$, координаты всех остальных пока задаем нулями (строки 5 и 6). Координаты первого жука мы продублируем, чтобы при прорисовке получать замкнутые многоугольники.

```
1  import matplotlib.pyplot as plt
2  from math import pi, sin, cos
3  n = 6
4  a = 2 * pi / n
5  X = [1] + [0] * n
6  Y = [0] + [0] * n
7  for t in range(60):
8      for k in range(1, n + 1):
9          X[k] = X[0] * cos(a * k) - Y[0] * sin(a * k)
10         Y[k] = X[0] * sin(a * k) + Y[0] * cos(a * k)
11     plt.plot(X, Y, c="skyblue")
12     X[0] += 0.15 * (X[1] - X[0])
13     Y[0] += 0.15 * (Y[1] - Y[0])
14 plt.gca().set_aspect("equal")
15 plt.show()
```

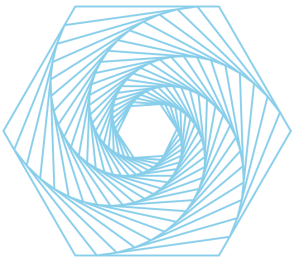


РИС. 8 Визуализация дискретной модели n жуков ($n = 6$)

Далее организуем цикл по времени (строки 7–13). На каждой его итерации сначала вычисляем координаты всех жуков, выполняя поворот координат первого жука на соответствующий угол (внутренний цикл по `k`). Далее соединяем линиями точки местоположения всех жуков с помощью команды `plot`, опция `c` задает цвет линии. Последним действием вычисляем новые координаты первого жука, сдвигая его по направлению к его соседу с коэффициентом $\delta = 0.15$. При выходе из цикла задаем одинаковый масштаб по обеим осям графика (аспектное отношение, строка 14) и выводим график на экран. Результат работы этого фрагмента кода для случая $n = 6$ показан на рис. 8.

ГЛАВА 5
Барометрическая формула

Задача • В покоящейся стационарной атмосфере давление равно отношению веса вышележащего столба воздуха к площади его поперечного сечения. Поэтому вполне очевидно, что с увеличением высоты давление воздуха должно уменьшаться. В самом деле, например, на высоте Эвереста давление составляет только одну четверть от давления на уровне моря.

Интересной и практически важной, например для авиации, задачей является определение точной зависимости давления воздуха P от высоты h над уровнем моря. В данной главе мы рассмотрим теоретическую модель стационарной изотермической атмосферы в гравитационном поле Земли.

Рассмотрим тонкий слой столба воздуха с площадью сечения S, заключенный между высотами h и $h+\Delta h$ (рис. 1). Так как атмосфера предполагается стационарной, то суммарная сила, действующая на выделенный слой воздуха, должна быть равна нулю. Запишем силы, действующие на этот объем в вертикальном направлении.

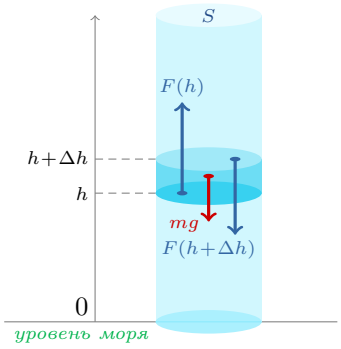


РИС. 1 Вертикальные силы, действующие на малый объем воздуха

Со стороны нижней части воздушного столба действует направленная вверх сила давления $F(h)$, равная произведению давления $P(h)$ на площадь сечения S. Со стороны верхней части этого же столба действует направленная вниз сила давления

$$F(h+\Delta h) = P(h+\Delta h)\cdot S.$$

Наконец, на выделенный слой действует направленная вниз сила тяжести mg, где m — масса этого слоя. Таким образом,

$$P(h+\Delta h)S + mg = P(h)S \Rightarrow \Delta P\cdot S = -mg, \qquad (1)$$

где $\Delta P = P(h + \Delta h) - P(h)$ — разность давлений на верхнем и нижнем основаниях нашего воздушного слоя. Домножим обе части последнего равенства на Δh и учтем, что $S\Delta h$ — это объем V данного слоя:

$$\Delta P \cdot V = -mg\Delta h. \tag{2}$$

Дмитрий Иванович Менделеев

Бенуа Поль Эмиль Клапейрон

Масса m в (2) зависит от высоты h (снизу воздух плотней, сверху — разреженней), ее можно найти с помощью закона Менделеева—Клапейрона, устанавливающего связь между давлением, объемом и температурой идеального газа[1]:

$$PV = \frac{mRT}{M}, \tag{3}$$

где $R = 8.314$ Дж/К·моль — универсальная газовая постоянная, T — температура газа в кельвинах (К), M — молярный объем, для воздуха $M = 0.029$ кг/моль. Выражаем из формулы (3) массу m, подставляем ее в (2) и после сокращения на V получаем следующее соотношение:

$$\Delta P = -\frac{MgP\Delta h}{RT}. \tag{4}$$

Делим обе части на Δh, делаем предельный переход $\Delta h \to 0$ и получаем искомое дифференциальное уравнение для неизвестной функции $P(h)$:

$$\frac{dP}{dh} = -\frac{Mg}{RT} \cdot P. \tag{5}$$

В качестве начального условия возьмем известное давление $P_0 = 101\,325$ Па на уровне моря:

$$P(0) = P_0. \tag{6}$$

В правой части уравнения (5) перед P имеется четыре коэффициента, два из которых (M и R) являются константами. Ускорение свободного падения g очень слабо меняется на высотах до 100 км[2], поэтому его тоже можно считать практически постоянным. А вот температура воздуха T, вообще говоря, зависит от высоты h[3]. В простейшем *изотермическом* случае температура воздуха предполагается постоянной величиной, не зависящей от высоты h. Соответствующий закон изменения давления от высоты, называемый теперь *барометрической формулой*, впервые сформулировал английский физик и астроном Галлей[4] в 1686

[1] Воздух при обычных условиях, т. е. при комнатной температуре и давлении на уровне моря, с хорошим приближением является идеальным газом.

[2] Например, на высоте 100 км ускорение свободного падения всего на 3 % меньше ускорения свободного падения у поверхности Земли.

[3] Известно, что до высоты 10 километров температура воздуха постепенно снижается со скоростью, примерно равной 6.5 К/км.

году, и только спустя более чем сто лет эта формула была уже математически выведена Лапласом. Как мы увидим дальше, барометрическая формула неплохо описывает давление воздуха на высотах до 3 километров.

Модель • Рассмотрим два варианта модели, основанных на дифференциальном уравнении (5). Первый вариант соответствует изотермическому случаю, когда температура атмосферы считается одинаковой по всей ее высоте. Второй, чуть более реалистичный вариант модели предполагает, что температура линейно падает с высотой. Полученные результаты моделирования сравним с экспериментальными данными.

1 Подключаем необходимые библиотеки. В этот раз помимо библиотеки SymPy нам потребуются (большей частью для построения графиков) еще две библиотеки — стандартная графическая библиотека Matplotlib и математический пакет NumPy.

```
from sympy import *
from IPython.display import display
import numpy as np
import matplotlib.pyplot as plt
```

2 Определяем все символы, входящие в дифференциальное уравнение (5) и начальное условие (6). Все эти величины, за исключением высоты h[5], являются положительными.

```
h = symbols("h")
M, g, R, T, P0 = symbols("M g R T P0",
                         positive = True)
P = Function("P")(h)
display(P)
```

$$P(h)$$

3 Составляем дифференциальное уравнение (5) и задаем начальное условие (6).

```
ode = Eq(diff(P, h), -M * g / (R * T) * P)
ic = {P.subs(h, 0): P0}
display(ode, ic)
```

$$\frac{d}{dh}P(h) = -\frac{MgP(h)}{RT}$$

```
{P(0): P0}
```

[4] Именем Галлея названа известная комета, возвращение которой в 1758 году в предсказанный Галлеем срок стало первым триумфальным подтверждением теории тяготения Ньютона.

Эдмунд Галлей

Пьер-Симон де Лаплас

[5] Отрицательная высота h означает, что мы находимся ниже уровня моря.

4 Решаем поставленную начальную задачу.

```
dsol = dsolve(ode, ics = ic)
display(dsol)
```

$$P(h) = P_0 e^{-\frac{Mgh}{RT}}$$

Так как в процессе решения мы фактически предполагали, что все символы, входящие в уравнение, в том числе температура T, являются постоянными величинами, то найденная нами формула соответствует изотермическому случаю $T = \text{const}$. Именно эта формула и называется барометрической.

ТАБЛ. 1 Давление P (Па) на разных высотах h (м)

h	P
0	101 325
1 000	89 876
2 000	79 501
3 000	70 121
4 000	61 660
5 000	54 048
6 000	47 217
7 000	41 105
8 000	35 651
9 000	30 800
10 000	26 499
11 000	22 699
12 000	19 399
13 000	16 579
14 000	14 170
15 000	12 111
16 000	10 352
17 000	8 849
18 000	7 565
19 000	6 467
20 000	5 529

5 Сравним теперь предсказания нашей изотермической модели с экспериментально найденными значениями давления на разных высотах, приведенными в табл. 1. Для этого сформируем из двух столбцов данной таблицы два массива `NumPy`. Высота в таблице изменяется с постоянным шагом, поэтому соответствующий массив `dataH` заполним с помощью команды `arange`.

```
dataH = np.arange(0, 21000, 1000)
dataP = np.array([101325, 89876, 79501, ...])
print(dataH[:3], dataP[:3])
```

```
[0 1000 2000] [101325 89876 79501]
```

6 Зададим параметры модели, соответствующие земной атмосфере. В качестве давления P_0 на уровне моря возьмем первый элемент массива `dataP`. Подставим эти параметры в решение `dsol`.

```
par = {R: 8.314, M: 0.029, g: 9.8, P0: dataP[0]}
ds = dsol.rhs.subs(par)
display(ds)
```

$$101325 e^{-\frac{0.0341833052682223h}{T}}$$

Получили формулу, выражающую давление P через высоту h и (постоянную) температуру T.

7 Чрезвычайно удобным инструментом библиотеки `SymPy` является команда `lambdify`, преобразующая заданное символьное выражение `SymPy` в обычную функцию `Python`. Первым аргументом команды `lambdify` является переменная или список переменных, от которых зависит заданное выражение, вторым аргументом

служит само выражение. Результатом работы этой команды является анонимная функция[6], зависящая от указанных параметров, которую можно сохранить в какой-нибудь переменной и использовать в дальнейшем как обычную функцию Python. Преобразуем с помощью этой команды правую часть выражения ds в функцию fP двух переменных h и T и, для примера, выведем на печать значение давления на высоте 10 км при температуре воздуха $T = 253\,\text{K}$ ($-20\,^\circ\text{C}$).

[6] Отсюда и происходит название команды.

```
fP = lambdify([h, T], ds)
print(int(fP(10000, 253)))
```

```
26238
```

Видно, что найденная величина незначительно отличается от соответствующего значения 26 499 в табл. 1.

8 Функции, генерируемые командой lambdify, умеют работать с массивами NumPy, поэтому мы можем применить функцию fP сразу ко всем высотам из массива dataH. Вторым аргументом для fP укажем температуру $T = 273\,\text{K}$. Результатом будет новый массив NumPy с предсказанными моделью значениями давления. Запомним этот массив в переменной model. Выведем на экран его первые три элемента, приведенные к целочисленному типу.

```
model = fP(dataH, 273)
print(model[:3].astype(int))
```

```
[101325 89399 78878]
```

9 Теперь мы можем визуализировать данные из массива model и сравнить их с экспериментальными данными. Для построения графиков используем команду plot библиотеки Matplotlib. Первым аргументом этой команды указывается массив x-координат точек, вторым — массив y-координат тех же точек (массивы должны иметь одинаковую длину), третий аргумент определяет вид кривой (этот аргумент необязательный, по умолчанию график строится в виде ломаной линии, соединяющей заданные точки). Остальные параметры являются опциональными, например параметр label определяет метку для легенды, параметр c задает цвет кривой. Первым графиком будет график нашей модели, x-координаты точек берем из массива

dataH, y-координаты — из массива model. Второй график соответствует экспериментальным данным (массивы dataH и dataP), которые мы изобразим с помощью маркеров в форме кружков (за это отвечает третий аргумент "o" команды plot в строке 2). В четвертой строке создаем легенду. Последней строкой выводим на экран построенный график (рис. 2).

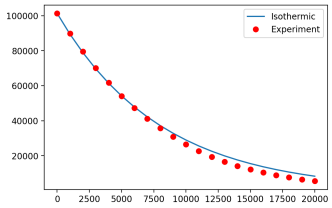


РИС. 2 Сравнение изотермической модели с экспериментальными данными

```
1 plt.plot(dataH, model, label = "Isothermic")
2 plt.plot(dataH, dataP, "o", c = "red",
3          label = "Experiment")
4 plt.legend()
5 plt.show()
```

10 Видно, что две кривые демонстрируют хорошее совпадение на небольших высотах, после чего они начинают расходиться. Переведем это наблюдение в числа, посчитав относительную погрешность (в процентах) модельных данных по формуле

$$E(h) = \left| \frac{P_M(h) - P(h)}{P(h)} \right| \cdot 100\,\%, \tag{7}$$

где $P_M(h)$ — давление на высоте h согласно модели, $P(h)$ — экспериментальное значение давления на той же высоте. Применим эту формулу сразу ко всему содержимому массивов model и dataP, на выходе получим новый массив погрешностей, который сохраним в переменной error. Построим график погешности (рис. 3), который и подтверждает сделанное нами наблюдение о хорошем совпадении модели и эксперимента на высотах до 5 км (ошибка менее одного процента). Однако на больших высотах погрешность начинает возрастать, и на высоте 20 км ошибка составляет уже порядка 50 %.

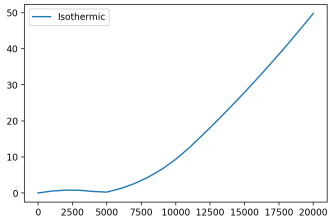


РИС. 3 Относительная погрешность изотермической модели

```
1 error = np.abs((model - dataP) / dataP) * 100
2 plt.plot(dataH, error, label = "Isothermic")
3 plt.legend()
4 plt.show()
```

11 Как уже упоминалось выше, на высотах до 10 км температура воздуха постепенно снижается. С хорошим приближением можно считать, что

$$T(h) = 288 - 0.0064h, \tag{8}$$

где h измеряется в метрах, а температура T — в градусах Кельвина. Построим в SymPy такую модель с линейной зависимостью температуры воздуха от высо-

ты. Введем два дополнительных символа a и b и подставим в исходное дифференциальное уравнение `ode`[7] вместо символа `T` выражение `a + b * h`.

```
a, b = symbols("a b")
ode2 = ode.subs(T, a + b * h)
display(ode2)
```

$$\frac{d}{dh}P(h) = -\frac{MgP(h)}{R(a+bh)}$$

[7] Заменять T на функцию от h надо именно в дифференциальном уравнении, а не в его решении!

12 Решаем начальную задачу для нового уравнения `ode2` со старым начальным условием `ic`.

```
dsol2 = dsolve(ode2, P, ics = ic)
display(dsol2)
```

$$P(h) = P_0 e^{\frac{Mg(-\log(R)-\log(a+bh))}{Rb}} e^{\frac{Mg\log(R)}{Rb}} e^{\frac{Mg\log(a)}{Rb}}$$

[8] Символ `T` из новой модели нами исключен.

13 Далее повторяем шаги 6–10 для новой модели. Подставляем параметры модели в решение, добавив предварительно параметры

$$a = 288 \text{ и } b = -0.0064$$

к словарю `par`, результат подстановки запоминаем в переменной `ds2`. Преобразуем правую часть решения в функцию `fP2`, зависящую теперь от единственной переменной `h`[8]. Применяем эту функцию к массиву `dataH`, результат записываем в массив `model2`. Строим график новой зависимости давления от высоты, указав метку `"Linear"` (см. рис. 4). Далее вычисляем массив погрешностей `error2` и строим соответствующий график (рис. 5). Предсказания новой модели с линейным убыванием температуры на первом графике теперь визуально практически не отличимы от экспериментальных данных, второй график с погрешностью модели подтверждает это наблюдение. Видно, что новая модель очень хорошо согласуется с реальными данными на высотах до 10–12 км, после чего погрешность начинает возрастать. Этот график, кстати, служит косвенным подтверждением того, что на высоте около 10 км закон изменения температуры становится каким-то другим (см. упражнение 7).

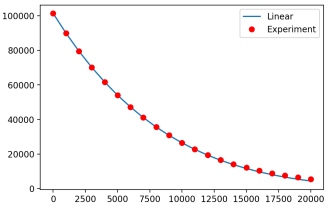


РИС. 4 Сравнение модели с линейным убыванием температуры с экспериментальными данными

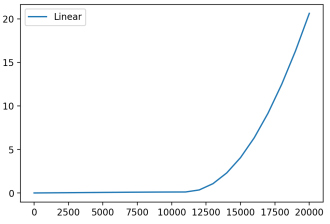


РИС. 5 Относительная погрешность модели с линейным убыванием температуры

14 Для более наглядного сравнения двух построенных нами моделей объединим их графики (рис. 6).

```
plt.plot(dataH, model, label = "Isothermic")
plt.plot(dataH, model2, label = "Linear")
plt.plot(dataH, dataP, "o", c = "red",
         label = "Experiment")
plt.legend()
plt.show()
```

15 Аналогичным образом объединяем графики с погрешностями двух моделей (рис. 7).

```
plt.plot(dataH, error, label = "Isothermic")
plt.plot(dataH, error2, label = "Linear")
plt.legend()
plt.show()
```

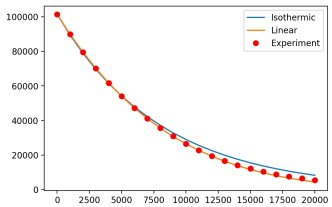


РИС. 6 Сравнение двух моделей друг с другом и с экспериментальными данными

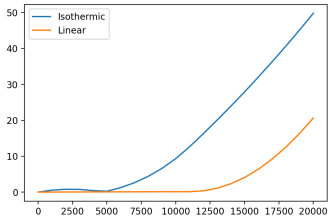


РИС. 7 Сравнение относительной погрешностей двух моделей

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Постройте графики зависимости давления P от высоты h в изотермической модели, используя разные значения температуры T. Сравните между собой погрешности этих моделей.

2 Характеристической высотой изотермической модели, показывающей скорость изменения давления с высотой, является высота $h_{1/2}$, на которой давление составляет половину давления на уровне моря. Вычислите значение такой высоты для атмосферы Земли.

3 Полученные нами формулы справедливы и для отрицательных высот. Оцените с их помощью давление воздуха на глубине 1 км. На какой глубине давление воздуха превысит давление на поверхности P_0 ровно в два раза?

4 Используя закон Менделеева—Клайперона (3) и решение `dsol`, найденное для изотермического случая, выразите плотность $\rho = m/V$ воздуха через высоту h. Зная зависимость $\rho(h)$, оцените примерную массу атмосферы Земли, посчитав тройной интеграл по всему объему атмосферы, т. е. по внешней части сферы радиуса, равного радиусу Земли. В сферических координатах этот интеграл имеет следующий вид:

$$m_A = \int_0^{2\pi} d\varphi \int_{-\pi/2}^{\pi/2} \cos\theta d\theta \int_{R_E}^{\infty} \rho(r - R_E) \cdot r^2 dr, \qquad (9)$$

где R_E — радиус Земли, $r - R_E = h$ — высота над уровнем моря (над поверхностью Земли).

5 Найдите параметры атмосфер других планет Солнечной системы — молярную массу M, температуру T и давление P_0 на уровне поверхности[9] — и постройте зависимость атмосферного давления от высоты для этих планет в изотермическом приближении.

[9] Также вам потребуются значения ускорения свободного падения на поверхности данных планет.

6 Предположим, в далеком будущем человечество собирается колонизировать Луну, создав на ней искусственную атмосферу, химически идентичную земной атмосфере. На уровне поверхности Луны атмосферное давление должно быть равным давлению P_0 земной атмосферы на уровне моря, а температура $T = 20\,^\circ\mathrm{C}$. Постройте зависимость давления такой искусственной атмосферы от высоты. На какой высоте давление будет в два раза меньше давления P_0?

7 После 10 км температура воздуха перестает понижаться и до высоты 25 км остается примерно на одном и том же уровне[10]. То есть на этих высотах атмосфера является изотермической. Определите зависимость $P(h)$ для такой кусочно-линейной температуры $T(h)$. Для этого надо с помощью решения `dsol2` нашей второй модели вычислить давление $P_1 = P(10\,000)$ на высоте 10 км и использовать это значение в качестве начального условия для дифференциального уравнения `ode` в изотермическом случае. Сравните полученное решение на высотах от 10 до 20 км с экспериментальными данными.

[10] Тропопауза и нижний слой стратосферы.

8 Составьте и решите дифференциальное уравнение для случая изотермической атмосферы в предположении, что ускорение свободного падения зависит от высоты h следующим образом:

$$g(h) = \frac{GM_E}{(R_E + h)^2}, \tag{10}$$

где $G = 6.67408 \cdot 10^{11}$ м^3·с$^{-2}$·кг$^{-1}$ — гравитационная постоянная, $M_E \approx 6 \cdot 10^{24}$ кг — масса Земли, $R_E \approx 6.4 \cdot 10^6$ м — ее радиус.

9 Постройте модель для вычисления давления жидкости на глубине h в предположении, что на ее поверхности давление равно P_0. При построении такой модели надо учесть, что жидкость в отличие от газа является несжимаемой, т. е. ее плотность *не меняется* с глубиной. Выведите и решите дифференциальное уравнение для этой модели. Оцените с помощью найденного решения давление воды на глубине 10 м, если давление на ее поверхности равно атмосферному давлению.

10 Устройство простейшего ртутного барометра (рис. 8) основано на приниципе уравновешивания столба воздуха и столба жидкости (в данном случае ртути). В стеклянную трубку, закрытую с одной стороны, наливается ртуть, трубка переворачивается и помещается в кювету со ртутью. Часть ртути из трубки выливается в кювету, из-за чего в верхней части трубки образуется ваккуум. Давление ртути на ее уровне в кювете равно атмосферному давлению. С другой стороны такое же давление создает ртуть,

РИС. 8 Схема простейшего ртутного барометра

оставшаяся в трубке. Высота ртути в трубке относительно высоты в кювете и служит мерой давления. Вычислите с помощью модели из предыдущего упражнения высоту ртутного столба для нормального атмосферного давления $P_0 = 101\,325$ Па. Оцените аналогичную высоту столба воды в простейшем водяном барометре.

11 Оценить значение некоторого выражения SymPy при заданных значениях входящих в него переменных можно с помощью комбинации методов `subs` и `evalf`:

```
f = x * sin(x)
display(f.subs(x, 2).evalf())
```

```
1.81859485365136
```

Однако такой способ является весьма неэффективным, если применять его к одному и тому же выражению многократно[11], т. к. каждое одиночное оценивание выражения сводится к выполнению серии относительно трудоемких символьных преобразований. Существенно более эффективным и более наглядным является в данном случае использование команды `lambdify`:

[11] Например, при построении графика.

```
F = lambdify(x, f)
print(F(2))
```

```
1.81859485365136
```

Исследуйте и сравните время работы этих двух подходов в зависимости от числа n оцениваний заданного выражения.

12 Дифференциальные уравнения вида $p(x)y' + q(x)y = 0$ называются *линейными однородными*. Решите с помощью SymPy следующие уравнения данного типа:

1) $y' + 4y = 0$;
2) $xy' = 2y$;
3) $y' + \operatorname{tg} x\, y = 0$;
4) $x \ln x\, y' = 2y$.

Постройте с помощью `Matplotlib` графики семейств найденных общих решений.

ГЛАВА 6
Модели роста

Модель естественного роста • Богатым источником дифференциальных уравнений является *популяционная экология*, особенно в той ее части, где исследуется динамика различных популяционных процессов. Базовой и исторически первой моделью популяционной динамики является так называемая модель естественного, или экспоненциального, роста. Впервые эта модель была предложена в 1798 году английским священником и ученым Томасом Мальтусом применительно к росту численности народонаселения.

Томас Мальтус

Пусть у нас имеется популяция некоторых живых организмов — бактерий, мух, кроликов, людей и т. п. Размер популяции x предполагается достаточно большим, чтобы его можно было считать непрерывной величиной. Предположим, что в единицу времени на одну особь популяции приходится α рождений новых особей[1] и β смертей имеющихся особей. Параметры α и β в модели естественного роста являются постоянными величинами. Требуется определить, как изменяется со временем размер такой популяции $x(t)$.

[1] Например, если $\alpha = 0.1$, то в популяции из миллиона особей в указанную единицу времени рождается примерно 100 тысяч новых особей.

Уравнение, описывающее динамику размера популяции, составляется по стандартной (*бухгалтерской*) схеме: рассмотрим, как изменяется число особей в популяции за промежуток времени от t до $t + \Delta t$:

$$x(t + \Delta t) = \underbrace{x(t)}_{\text{было}} + \underbrace{\alpha \cdot x(t) \cdot \Delta t}_{\text{родилось}} - \underbrace{\beta \cdot x(t) \cdot \Delta t}_{\text{умерло}}. \qquad (1)$$

Следовательно,

$$\frac{\Delta x}{\Delta t} = (\alpha - \beta) \cdot x(t). \qquad (2)$$

После выполнения предельного перехода $\Delta t \to 0$ по-

лучаем искомое дифференциальное уравнение:

$$\frac{dx}{dt} = kx, \tag{3}$$

где $k = \alpha - \beta$ — удельная скорость роста популяции. В качестве начального условия традиционно используется известный размер популяции в начальный момент времени (обычно при $t = 0$):

$$x(0) = x_0. \tag{4}$$

Модель Ферхюльста • Модель Мальтуса, рассмотренная нами выше, считается адекватной только для начальных стадий роста популяции, т. е. при условии неограниченности ресурсов и отсутствия внутривидового соперничества за эти ресурсы. Так как в любой реальной популяции количество ресурсов является конечным, то начиная с некоторого момента внутри популяции начинается борьба за эти ресурсы, что приводит к уменьшению удельной скорости роста популяции.

Пьер Франсуа Ферхюльст

Первой моделью, в которой учитывалась внутривидовая конкуренция за ресурсы, была модель *логистического роста*, предложенная в 1838 году бельгийским математиком Пьером Ферхюльстом[2]. В модели Ферхюльста предполагается, что скорость изменения размера популяции $\dot{x}$ пропорциональна самому этому размеру x (как и в модели Мальтуса), а также пропорциональна количеству доступных популяции ресурсов. При этом количество ресурсов должно убывать с ростом популяции.

[2] Историки науки так и не установили, почему Ферхюльст назвал свой закон «логистическим».

В простейшем случае число ресурсов представляет собой убывающую линейную функцию от размера популяции. При таком предположении дифференциальное уравнение, описывающее рост популяции в модели Ферхюльста, записывается следующим образом:

$$\frac{dx}{dt} = kx\left(1 - \frac{x}{M}\right), \tag{5}$$

где $k > 0$ — удельная скорость размножения, M — так называемая *поддерживающая емкость среды*.

Модель • Построим в `SymPy` описанные выше модели роста популяции — модель естественного роста (или

модель Мальтуса) и логистическую модель Ферхюльста. Используя полученные решения, попробуем применить эти две модели для оценки численности населения Земли, начиная с 1950 года.

1 Подключаем необходимые библиотеки, в том числе библиотеку `Matplotlib` и пакет `NumPy`.

2 Определяем символы для переменной t и двух параметров k и x_0, а также функцию $x(t)$.

3 Создаем дифференциальное уравнение (3) для модели естественного роста.

```
ode = Eq(diff(x, t), k * x)
display(ode)
```

$$\frac{d}{dt}x(t) = kx(t)$$

4 Определяем начальное условие (4).

```
ic = {x.subs(t, 0): x0}
display(ic)
```

```
{x(0): x0}
```

5 Решаем уравнение `ode` с начальным условием `ic`. Решение запоминаем в переменной `dsol`.

```
dsol = dsolve(ode, x, ics = ic)
display(dsol)
```

$$x(t) = x_0 e^{kt}$$

6 Найденное решение зависит от двух параметров — начального размера популяции x_0 и удельной скорости роста k. Исследуем графически характер этих зависимостей. Сначала построим семейство интегральных кривых для разных значений x_0 при фиксированной положительной скорости $k = 0.04$ (рис. 1).

```
p1 = plot(ylim = (0, 40), show = False)
for xx in np.arange(1, 40, 2):
    ds = dsol.rhs.subs({k: 0.04, x0: xx})
    p1.extend(plot(ds, (t, 0, 100), show = False))
p1.show()
```

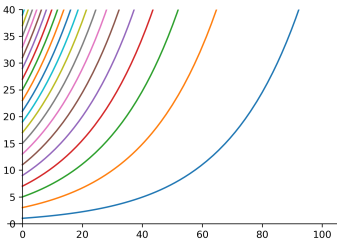


РИС. 1 Поведение модели естественного роста для разных значений параметра x_0

7 Далее построим семейство интегральных кривых уже для разных значений k (в том числе и отрицательных) при фиксированной начальной численности $x_0 = 15$ (рис. 2).

```
p2 = plot(ylim = (0, 40), show = False)
for kk in np.arange(-0.2, 0.2, 0.01):
    ds = dsol.rhs.subs({k: kk, x0: 15})
    p2.extend(plot(ds, (t, 0, 100), show = False))
p2.show()
```

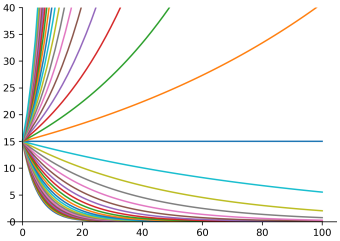


РИС. 2 Поведение модели естественного роста для разных значений параметра k

Убывающие экспоненты на графике соответствуют отрицательным значениям k (популяция вымирает), возрастающие — положительным значениям k (популяция неограниченно растет). Критическим значением является $k = 0$ (горизонтальная прямая $x = x_0$), при котором размер популяции не меняется со временем.

8 Попробуем теперь применить построенную модель к данным ООН по численности населения Земли с 1950 по 2020 год (см. табл. 1). Для этого создадим и заполним два массива `NumPy` данными из приведенной таблицы.

ТАБЛ. 1 Численность населения Земли (в млрд человек) в разные годы

Год	Население
1950	2.53
1955	2.76
1960	3.02
1965	3.33
1970	3.69
1975	4.06
1980	4.44
1985	4.85
1990	5.29
1995	5.71
2000	6.11
2005	6.51
2010	6.91
2015	7.30
2020	7.67

```
years = np.arange(1950, 2021, 5)
pop = np.array([2.53, 2.76, 3.02, 3.33, ...])
print(years[:3], pop[:3])
```

```
[1950 1955 1960] [2.53 2.76 3.02]
```

9 Будем отсчитывать время ($t = 0$) в нашей модели с 1950 года. Понятно, что параметр x_0 — это численность населения Земли в 1950 году, т. е. вместо параметра x_0 мы должны подставить в решение дифференциального уравнения величину `pop[0]`. Немного сложнее дело обстоит со вторым параметром модели k, значения которого в явном виде в наших данных нет. Если предположить, что данные в табл. 1 являются точными и соответствующими модели естественного роста, то параметр k можно оценить по результатам ровно одного измерения x_1 в любой момент времени t_1, не совпадающий с начальным моментом t_0. Введем два дополнительных символа для параметров x_1 и t_1 и подставим их в решение дифференциального уравнения `dsol` вместо `x` и `t`. Получим алгебраическое уравнение `eq` относительно символа `k`.

```
t1, x1 = symbols("t1 x1")
eq = dsol.subs({x: x1, t: t1})
display(eq)
```

$$x_1 = x_0 e^{kt_1}$$

10 Для решения полученного уравнения `eq` применим команду `solve`, которая возвращает в качестве результата список с найденными решениями[3]. В данном случае этот список состоит из единственного решения, извлечем его с помощью нулевого индекса и сохраним в переменной `sol`.

```
sol = solve(eq, k)[0]
display(sol)
```

$$\frac{\log\left(\frac{x_1}{x_0}\right)}{t_1}$$

11 Теперь можем подставить найденное выражение вместо символа `k` в решение `dsol`.

```
dsol2 = dsol.subs(k, sol)
display(dsol2)
```

$$x(t) = x_0 e^{\frac{t \log\left(\frac{x_1}{x_0}\right)}{t_1}}$$

Получили решение уравнения, выраженное через три параметра x_0, x_1 и t_1, значения которых мы можем взять, например, из массивов `pop` и `years`.

12 Построим с помощью `Matplotlib` график полученной модели и сравним его с реальными данными из табл. 1. Выберем в качестве величины t_1 значение 30 лет, что соответствует 1980 году, в массиве `pop` нужное нам значение x_1 хранится под индексом $30/5 = 6$. Создаем словарь `par` с параметрами модели. Преобразуем правую часть решения с подставленными в нее параметрами в функцию `Python`. Вычисляем предсказание `prediction` модели для каждого года из массива `years`, т. к. в модели отсчет времени ведется с нуля, то вычитаем из каждого года величину 1950. Строим график с предсказанием и для сравнения график с исходными данными модели. Включаем показ легенды и выводим график на экран.

```
par = {x0: pop[0], t1: 30, x1: pop[6]}
fP = lambdify(t, dsol2.rhs.subs(par))
prediction = fP(years - 1950)
plt.plot(years, prediction, label = "1980")
plt.plot(years, pop, "o", label = "ООН")
plt.legend()
plt.show()
```

[3] Команда `solve` считается устаревшей командой `SymPy`, от которой разработчики библиотеки планируют в будущем отказаться. См. замечение 1 к главе для объяснения, почему мы не использовали для решения данного алгебраического уравнения стандартный решатель `solvesest`.

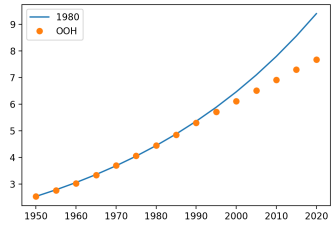


РИС. 3 Сравнение модели Мальтуса с реальными данными

Результат сравнения показан на рис. 3. Видно, что модель показывает адекватные результаты только на датах примерно до 1990 года, после чего предсказания модели начинают существенно расходиться с реальными данными, т. е. рост населения Земли не подчиняется построенной нами модели Мальтуса.

13 Построим теперь модель Ферхюльста по такой же схеме. Определяем дополнительный символ `M` и составляем дифференциальное уравнение (5).

```
M = symbols("M")
ode2 = Eq(diff(x, t), k * x * (1 - x / M))
display(ode2)
```

$$\frac{d}{dt}x(t) = k\left(1 - \frac{x(t)}{M}\right)x(t)$$

14 Решаем уравнение, используя начальное условие из модели Мальтуса. Упрощаем результат с помощью команды `symplify`[4].

```
dsol3 = simplify(dsolve(ode2, x, ics = ic))
display(dsol3)
```

$$x(t) = \frac{Mx_0e^{kt}}{M + x_0e^{kt} - x_0}$$

[4] В неупрощенное решение нельзя подставить критическое значение $x_0 = M$, проверьте!

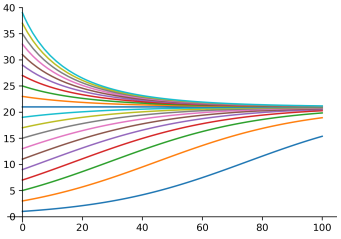


РИС. 4 Поведение модели Ферхюльста для разных значений параметра x_0

15 Построим семейство кривых найденного решения для разных значений x_0 при фиксированных остальных параметрах модели (см. рис. 4).

```
p3 = plot(ylim = (-1, 40), show = False)
for xx in np.arange(1, 40, 2):
    ds = dsol3.rhs.subs({k: 0.04, M: 21, x0: xx})
    p3.extend(plot(ds, (t, 0, 100), show = False))
p3.show()
```

Видно, что при любых начальных условиях численность популяции стремится к предельной величине, равной M. Кривые, стартующие ниже M, называются *сигмоидами*.

16 Применим построенную нами модель Ферхюльста к данным роста численности населения Земли. Теперь модель включает в себя два дополнительных параметра k и M, значения которых можно определить с помощью двух дополнительных условий $x_1 = x(t_1)$ и $x_2 = x(t_2)$. Получаемая при этом система двух нелинейных алгебраических уравнений относительно k и

M в случае произвольных t_1 и t_2, однако, не решается аналитически. Аналитическое решение возможно, например, при $t_2/t_1 = 2$, в этом случае система сводится к квадратному уравнению[5]. Создадим новый символ `x2` и подставим в решение `dsol3` условия $x_1 = x(t_1)$ и $x_2 = x(2t_1)$. Сохраним полученные уравнения в переменных `eq2` и `eq3`.

[5] При условии, что $t_0 = 0$. В общем случае условие разрешимости имеет вид

$$t_2 - t_1 = t_1 - t_0.$$

```
x2 = symbols("x2")
eq2 = dsol3.subs({x: x1, t: t1})
eq3 = dsol3.subs({x: x2, t: 2 * t1})
display(eq2, eq3)
```

$$x_1 = \frac{Mx_0e^{kt_1}}{M + x_0e^{kt_1} - x_0}$$
$$x_2 = \frac{Mx_0e^{2kt_1}}{M + x_0e^{2kt_1} - x_0}$$

17 Решим систему из этих двух уравнений с помощью команды `solve`[6]. Первым аргументом этой команде в данном случае надо передать кортеж из двух уравнений, вторым — кортеж из символов неизвестных. Решением является список кортежей. В нашем случае решение одно, извлекаем его с помощью нулевого индекса и распаковываем в переменные `sol2` и `sol3`.

[6] См. замечание 2 к главе.

```
sol2, sol3 = solve((eq2, eq3), (k, M))[0]
display(sol2, sol3)
```

$$\frac{\log\left(\frac{x_2(x_0-x_1)}{x_0(x_1-x_2)}\right)}{t_1}$$
$$\frac{x_1\left(-x_0x_1 + 2x_0x_2 - x_1x_2\right)}{x_0x_2 - x_1^2}$$

18 Подставляем эти два выражения вместо символов `k` и `M` в решение `dsol3` дифференциального уравнения. Для вывода на экран полученное выражение упрощаем с помощью команды `logcombine`, которая в данном случае вносит все коэффициенты, стоящие перед логарифмами, под логарифмы как степени (опция `force=True`) и упрощает получающиеся при этом экспоненты.

```
dsol4 = dsol3.subs({k: sol2, M: sol3})
display(logcombine(dsol4, force = True))
```

$$x(t) = \frac{x_0 x_1 \left(\frac{x_2(x_0-x_1)}{x_0(x_1-x_2)}\right)^{\frac{t}{t_1}} (-x_0x_1+2x_0x_2-x_1x_2)}{(x_0x_2-x_1^2)\left(x_0\left(\frac{x_2(x_0-x_1)}{x_0(x_1-x_2)}\right)^{\frac{t}{t_1}} - x_0 + \frac{x_1(-x_0x_1+2x_0x_2-x_1x_2)}{x_0x_2-x_1^2}\right)}$$

Получили в итоге решение $x(t)$, выраженное через параметры x_0, x_1, x_2 и t_1.

19 Применим построенную модель к предсказанию роста населения Земли. Как и в предыдущей модели, положим $t_1 = 30$ (1980 год), следовательно, $t_2 = 60$, что соответствует 2010 году. Соответствующие данные в массиве `pop` хранятся под индексами 6 и 12. Все остальные действия аналогичны пункту 12.

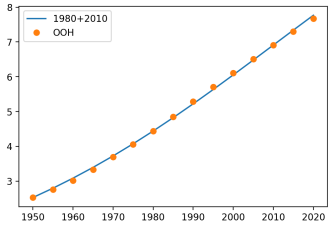


РИС. 5 Сравнение модели Ферхюльста с реальными данными

```
par = {x0: pop[0], t1: 30, x1: pop[6], x2: pop[12]}
fP2 = lambdify(t, dsol4.rhs.subs(par))
prediction = fP2(years - 1950)
plt.plot(years, prediction, label = "1980+2010")
plt.plot(years, pop, "o", label = "ООН")
plt.legend()
plt.show()
```

20 Наконец, оценим величину поддерживающей емкости среды M согласно последней построенной модели. В пункте 17 мы выразили этот параметр модели через величины x_0, x_1, x_2 и t_1 и сохранили полученное выражение в переменной `sol3`. Подставим в это выражение числовые параметры `par` последней модели.

```
print(sol3.subs(par))
```

```
13.8276161878725
```

Так как поддерживающая емкость среды представляет собой предельное значение размера популяции в модели Ферхюльста, то получаем, что *согласно построенной модели* предельная численность населения Земли составляет примерно 14 миллиардов.

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Если мы попытаемся решить уравнения `eq` из пункта 10 с помощью команды `solveset`, то получим следующий неприятный результат, обусловленный тем, что `SymPy` по умолчанию все решает в комплексных числах:

```
solk = solveset(eq, k)
display(solk)
```

$$\left\{ \frac{i\left(2n\pi + \arg\left(\frac{x_1}{x_0}\right)\right) + \log\left(\left|\frac{x_1}{x_0}\right|\right)}{t_1} \;\middle|\; n \in \mathbb{Z} \right\}$$

Чтобы найти решение в действительных числах, надо указать опцию `domain = S.Reals`:

```
solk = solveset(eq, k, domain = S.Reals)
display(solk)
```

$$\mathbb{R} \cap \left\{ \frac{\log\left(\frac{x_1}{x_0}\right)}{t_1} \right\}$$

Из такого выражения уже можно извлечь собственно решение, используя дважды атрибут `args`:

```
display(solk.args[1].args[0])
```

$$\frac{\log\left(\frac{x_1}{x_0}\right)}{t_1}$$

2 Для решения систем нелинейных уравнений в SymPy имеется специальная команда `nonlinsolve`. Однако для решения системы уравнений `(eq2, eq2)` (см. пункт 16) эта команда не работает. Данную систему можно решить в *ручном режиме* с помощью нескольких применений команды `solveset`. Сначала разрешаем каждое из двух уравнений относительно `M` и приравниваем найденные выражения:

```
m1 = solveset(eq2, M).args[0].args[0]
m2 = solveset(eq3, M).args[0].args[0]
eqk = Eq(m1, m2)
display(eqk)
```

$$\frac{x_0 x_1 \left(e^{kt_1} - 1\right)}{x_0 e^{kt_1} - x_1} = \frac{x_0 x_2 \left(e^{kt_1} - 1\right)\left(e^{kt_1} + 1\right)}{x_0 e^{2kt_1} - x_2}$$

В полученном уравнении делаем замену $e^{kt_1} \to K$:

```
K = symbols("K")
eqK = eqk.subs(k, log(K) / t1)
display(eqK)
```

$$\frac{x_0 x_1 \left(K - 1\right)}{K x_0 - x_1} = \frac{x_0 x_2 \left(K - 1\right)\left(K + 1\right)}{K^2 x_0 - x_2}$$

Это уравнение уже решается с помощью `solveset`:

```
solK = solveset(eqK, K).args[0].args[2]
display(solK)
```

$$\frac{x_0x_2 - x_1x_2}{x_0x_1 - x_0x_2}$$

Делаем обратную замену и подставляем полученное выражение вместо символа `k` в одно из выражений `m1` или `m2`. Это и дает нам искомое решение системы:

```
solk = log(solK) / t1
solM = m1.subs(k, solk)
display(solk, simplify(solM))
```

$$\frac{\log\left(\frac{x_0x_2 - x_1x_2}{x_0x_1 - x_0x_2}\right)}{t_1}$$

$$\frac{x_1\left(-x_0x_1 + 2x_0x_2 - x_1x_2\right)}{x_0x_2 - x_1^2}$$

3 По аналогии с пунктом 12 постройте на одном графике предсказания численности населения Земли с помощью модели Мальтуса с дополнительным условием $x(t_1) = x_1$ для значений времени $t_1 = 5, 10, \ldots, 50$.

4 Постройте графики предсказания численности населения Земли с помощью модели Ферхюльста для разных значений времени $t_1 = 5, 10, \ldots, 35$. Для каждого из этих значений оцените предельную численность M.

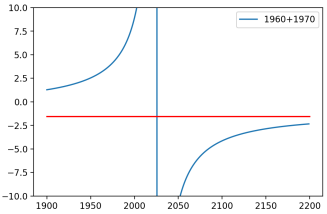
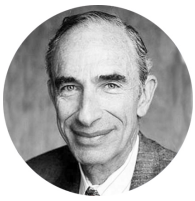


РИС. 6 Конец света в модели Ферхюльста

5 Если мы оценим параметры построенной в главе модели Ферхюльста для населения Земли, например, для случая $t_1 = 10$ (т. е. по данным 1960 и 1970 годов), то обнаружим, что поддерживающая емкость среды M является отрицательной. Это значит, что экспериментальные данные и модель не соответствуют друг другу. Если построить график найденной зависимости $x(t)$ в более более широком диапазоне (см. рис. 6, красная линия соответствует отрицательной предельной численности населения), то мы увидим, что у этого графика имеется разрыв примерно в 2025 году. При приближении к этой дате модель предсказывает бесконечный рост населения Земли, что является абсолютно нереалистичным. Такой виртуальный апокалипсис в теории популяционного моделирования с помощью дифференциальных уравнений называется *doomsday* (дословно: конец света).



Пол Ральф Эрлих

6 По оценкам известного демографа Пола Эрлиха, 10 тысяч лет назад на Земле жило около 5 млн человек. В 2000 году — примерно 6 млрд человек. Оцените, исходя из этих данных, примерное время «возникновения» человечества согласно модели Мальтуса, т. е. когда его численность составляла ровно 2 человека. Попробуйте оценить это же время с помощью данных из табл. 1.

7 Найдите в Интернете данные по численности населения по континентам и отдельным странам и постройте соответствующие модели Мальтуса и Ферхюльста.

8 Если проанализировать решение дифференциального уравнения

$$x(t) = \frac{Mx_0e^{kt}}{M + x_0e^{kt} - x_0} \tag{6}$$

для модели Ферхюльста, то окажется, что существует такая функция $X(t)$, что $x(t) < X(t)$ для любого начального значения $x_0 > 0$ (см. рис. 7, функция $X(t)$ выделена красным цветом). Найдите эту функцию, используя команду `limit`.

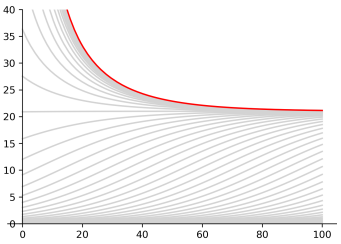


РИС. 7 Предельное решение в модели Ферхюльста

9 Постройте семейства графиков решения $x(t)$ дифференциального уравнения в модели Ферхюльста для различных значений: а) параметра k; б) параметра M. Исследуйте поведение модели для отрицательных значений этих параметров.

10 Дифференциальное уравнение, возникающее в модели Мальтуса, принадлежит классу линейных однородных уравнений первого порядка вида

$$y' = f(x)y. \tag{7}$$

Известно, что общее решение этого уравнения имеет вид[7]:

$$y = Ce^{F(x)}, \text{ где } F(x) = \int f(x)dx. \tag{8}$$

[7] Эта формула элементарно выводится из решения данного уравнения методом разделения переменных.

Напишите функцию `lsolve(ode, y, x)`, которая бы решала с помощью формулы (8) заданное дифференциальное уравнение относительно заданной неизвестной функции: первым шагом разрешаем уравнение относительно производной — получаем выражение $f(x)y$; вторым шагом делим правую часть на неизвестную функцию — получаем функцию $f(x)$; последним шагом применяем формулу (8). Протестируйте написанную функцию на решении следующих уравнений:

1) $\dfrac{dy}{dx} - xy = 0$;
2) $\dfrac{t+1}{x}\dfrac{dx}{dt} = t$;
3) $\dfrac{dr}{d\varphi} = \dfrac{r(\cos\varphi + 1)}{\sin\varphi}$;
4) $(u^2 + 1)\dfrac{dv}{du} = uv$.

11 Дифференциальное уравнение в модели Ферхюльста относится к классу уравнений Бернулли:

$$\frac{dx}{dt} = p(t)x + q(t)x^n, \text{ где } n \neq 0, 1. \tag{9}$$

Библиотека SymPy не всегда классифицирует уравнения такого типа, например ею не классифицируется и не решается уравнение ($n = 100$)

$$\frac{dx}{dt} = x^{100} + x. \tag{10}$$

Стандартный алгоритм решения уравнений Бернулли заключается в замене неизвестной функции $x(t)$ на новую функцию $y(t)$, приводящей к линейному уравнению:

$$y = \frac{1}{x^{n-1}} \Rightarrow \frac{dy}{dt} = \frac{1-n}{x^n}\frac{dx}{dt} \Rightarrow$$
$$\Rightarrow \frac{1}{x^n}\frac{dx}{dt} = \frac{1}{1-n}\frac{dy}{dt} \Rightarrow \frac{1}{1-n}\frac{dy}{dt} = p(t)y + q(t). \tag{11}$$

Применим этот подход к решению уравнения (10):

```
1 n = 100
2 ode = Eq(diff(x, t), x + x ** n)
3 y = Function("y")(t)
4 zy = 1 / x ** (n - 1)
5 zx_ = x ** n * diff(y, t) / (1 - n)
6 rhs = solve(ode.subs(x_, zx_), diff(y, t))[0]
7 zode = Eq(diff(y, t), rhs.subs(zy, y))
8 dsolve(zode, y).subs(y, zy)
```

$$\frac{1}{x^{99}(t)} = C_1 e^{-99t} - 1$$

В первых трех строках явно задаем параметр n, определяем дифференциальное уравнение и вводим новую неизвестную функцию $y(t)$. Далее определяем замену и вычисляем ее производную по t. В строке 6 выполняем подстановку производной в исходное уравнение и разрешаем это уравнение относительно dy/dt. В полученном выражении завершаем замену и формируем новое дифференциальное уравнение. Решаем его и делаем обратную замену[8].

12 Убедитесь, что следующие дифференциальные уравнения являются уравнениями Бернулли, и решите их либо с помощью команды dsolve, либо по описанной выше схеме (определив предварительно значение n[9]):

1) $\frac{dx}{dt} + 2x = x^3$;
2) $t\frac{dx}{dt} + x = t^2x^{50}$;
3) $x^2\frac{dx}{dt} = 4x^3 + 1$;
4) $\frac{dx}{dt} = \left(x + \sqrt[5]{x}\right)\sin t$.

[8] Крайне рекомендуется вывести на экран результаты выполнения всех приведенных команд, начиная с четвертой, и сравнить их с формулами (11).

[9] Для дробных n используйте команду Rational.

ГЛАВА 7
Табулирование функций

Метод Эйлера • Далеко не все дифференциальные уравнения можно решить аналитически. Для решения таких «нерешаемых» уравнений[1] были придуманы специальные *приближенные* методы, позволяющие находить искомое решение с некоторой (управляемой) погрешностью. Простейшим приближенным методом решения дифференциальных уравнений является *метод Эйлера*.

Рассмотрим схему работы метода Эйлера решения начальной задачи

$$y' = f(x, y),\ y(x_0) = y_0.$$

Согласно определению производной[2] заменим в дифференциальном уравнении производную y' на разностное отношение

$$y'(x) \approx \frac{y(x+h) - y(x)}{h}, \qquad (1)$$

где h — параметр, называемый *шагом*:

$$\frac{y(x+h) - y(x)}{h} = f(x, y) \Rightarrow$$
$$\Rightarrow y(x+h) = y(x) + h \cdot f(x, y). \qquad (2)$$

Полученная формула позволяет вычислить неизвестную функцию в точке $x+h$, если известно ее значение в точке x. Подставляя в (2) значения из начального условия x_0 и y_0, находим

$$y_1 = y(x_0 + h) = y_0 + h \cdot f(x_0, y_0).$$

Подставляем в (2) $x_1 = x_0 + h$ и вычисленное значение y_1 и находим

$$y_2 = y(x_1 + h) = y_1 + h \cdot f(x_1, y_1).$$

[1] А также уравнений, аналитическое решение которых хотя и возможно, но слишком трудоемко.

Леонард Эйлер

[2] По определению производной

$$y'(x) = \lim_{h \to 0} \frac{y(x+h) - y(x)}{h},$$

следовательно, для малых h должно выполняться приближенное равенство

$$y'(x) \approx \frac{y(x+h) - y(x)}{h}.$$

Продолжая этот процесс, находим значения неизвестной функции y_i во всех точках вида $x_i = x_0 + ih$:

$$y_0 = y(x_0),\ y_{i+1} = y_i + h \cdot f(x_i, y_i),\ i = 0, 1, \ldots \qquad (3)$$

Геометрический смысл метода Эйлера вполне очевиден (см. рис. 1). Значение $k_1 = f(x_i, y_i)$ определяет угловой коэффициент (тангенс угла наклона) касательной к искомой кривой в текущей точке (x_i, y_i). Если мы придаем приращение аргументу $\Delta x = h$, то приращение функции Δy будет приблизительно равно $h \cdot k_1$, что и приводит нас к формуле (3).

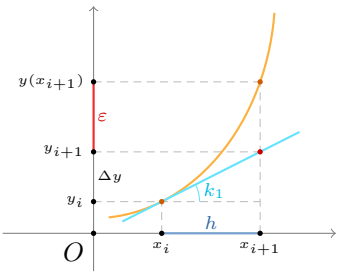
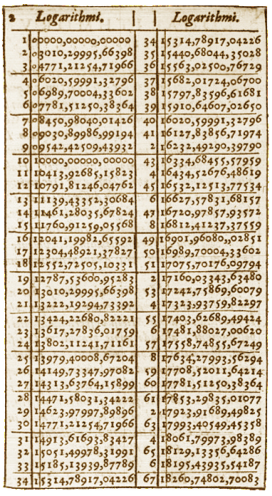


РИС. 1 Геометрическая интерпретация метода Эйлера


Logarithmi. | Logarithmi.

1	00000,00000,00000	34	45314,78917,04216
2	03010,29995,66398	35	54406,80443,35018
3	04771,21254,71966	36	55563,02500,76729
4	06020,59991,32796	37	56682,01724,06700
5	06989,70004,33602	38	57797,83596,61681
6	07781,51250,38364	39	58910,64607,02650
7	08450,98040,01416	40	60020,59991,32796
8	09030,89986,99194	41	61127,83856,71974
9	09542,42509,42932	2	62232,49290,39790
10	10000,00000,00000	43	63334,68455,57959
11	10413,92685,15823	4	64434,52076,48619
12	10791,81246,04761	45	65532,12513,77534
13	11139,43352,30684	6	66627,57831,68157
14	11461,28021,87824	47	67720,97857,93572
15	11760,91250,05568	8	68812,41237,37559
16	12041,19981,65592	49	69901,96080,02851
17	12304,48921,37827	50	69889,70004,33602
18	12552,72501,10321	51	70075,70176,09794
19	12787,53660,91283	2	71160,03343,63480
20	13010,29995,66398	53	72242,75869,60079
21	13222,19294,73392	4	73323,93759,82297
22	13424,22680,82221	55	74403,62689,49424
23	13617,27836,01759	6	74481,88027,00620
24	13802,11241,71161	57	75558,74855,67249
25	13979,40008,67204	8	76634,27993,56294
26	14149,73347,97082	59	77708,51011,64214
27	14313,63764,15899	60	77781,51250,38364
28	14471,58031,34422	61	78853,29855,01077
29	14623,97997,38986	2	79923,91689,49825
30	14771,21254,71966	63	79993,40549,45378
31	14913,61693,82437	4	81061,79973,98389
32	15051,49978,31981	65	82129,13356,64286
33	15185,13939,87789	6	82195,49235,54187
34	15314,78917,04216	67	83260,74801,70083

РИС. 2 Одна из первых таблиц десятичных логарифмов, 1617 г.


Чарльз Бэббидж

Задача • Табулированием функции называется составление таблицы ее значений в заданном диапазоне изменения ее аргумента. До появления калькуляторов и компьютеров, способных вычислять с нужной точностью тригонометрические, логарифмические и другие элементарные функции, таблицы были единственным средством практического использования таких функций. Например, тригонометрические функции активно применялись для навигации в мореплавании, логарифмические (рис. 2) — для ускорения арифметических вычислений (прежде всего для замены тяжелых операций умножения и деления на простые сложение и вычитание).

Составление таблиц функций — достаточно простая в алгоритмическом смысле задача, но требующая выполнения больших объемов чисто вычислительной работы, которая выполнялась вручную и приводила по этой причине к большому количеству арифметических ошибок. Поэтому еще с XVIII века предпринимались попытки по механизации и автоматизации такого рода деятельности, наиболее известной из которых может служить разностная машина Чарльза Бэббиджа, предложенная (но так до конца и не достроенная) им в 1822 году.

Проблема построения таблиц функций не потеряла своей актуальности и в настоящее время. Многие алгоритмы вычисления значений элементарных функций используют для ускорения расчетов таблицы предвычисленных значений этих функций. Кроме того, табулирование остается основным методом работы с многочисленными специальными функциями, поддержка которых не встроена в аппаратную часть современных компьютеров.

Рассмотрим подход к решению задачи табулирования функций, основанный на численном решении обыкновенных дифференциальных уравнений. Основная идея предлагаемого подхода заключается в том, что многие сложные функции (например, экспонента или логарифм) оказываются решениями начальных задач для относительно простых дифференциальных уравнений вида $y' = f(x, y)$, в которых правая часть является *рациональной* функцией, т. е. комбинацией четырех арифметических операций над переменными x и y. Например, функция $y = e^{x^2}$ является решением начальной задачи

$$y' = 2xy,\ y(0) = 1. \tag{4}$$

Следовательно, чтобы составить таблицу значений такой функции, достаточно приближенно решить, например методом Эйлера, соответствующую начальную задачу с заданным шагом h. Заметим, что если функция $f(x, y)$ рациональна, то все вычисления будут выполняться только с помощью четырех арифметических операций.

Модель • Рассмотрим решение задачи табулирования на примере функции ошибок $\operatorname{erf} x$, которая определяется как

$$\operatorname{erf} x = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt. \tag{5}$$

Известно, что эта функция является нечетной, поэтому достаточно построить таблицу ее значений только для положительных x. Составление таблицы мы будем выполнять в три этапа: вычисление константы π; вычисление квадратного корня из π; вычисление интеграла в (5). Каждый из этапов будет реализован через решение специально составленного дифференциального уравнения с рациональной правой частью с помощью метода Эйлера[3].

[3] Таким образом, для составления таблицы значений заданной функции мы не будем использовать ни математические функции, ни математические константы типа π и e, а только арифметику.

1 Подключаем библиотеки. В этот раз нам не понадобится библиотека символьных вычислений, а функции модуля `math` будут нужны только для проверки работоспособности нашей модели. Вычислим, для примера, значение $\operatorname{erf}(1)$.

```
import numpy as np
import matplotlib.pyplot as plt
import math
print(math.erf(1))
```

```
0.8427007929497149
```

2 Наш первый шаг — приближенное вычисление числа π. Заметим, что

$$\frac{\pi}{4} = \operatorname{arctg} 1 \Rightarrow \pi = y(1), \text{ где } y(x) = 4 \operatorname{arctg} x. \tag{6}$$

Нужно придумать дифференциальное уравнение с рациональной правой частью, решением которого была бы функция $y(x)$. В данном случае это сделать несложно, т. к. производная y' сама по себе является рациональным выражением:

$$y' = \frac{4}{1 + x^2}. \tag{7}$$

Общим решением этого уравнения выступает функция

$$y(x) = 4 \operatorname{arctg} x + C, \tag{8}$$

чтобы исключить из него параметр C, надо придумать подходящее начальное условие, при котором C обращается в ноль, например подойдет условие $y(0) = 0$. Решим с помощью метода Эйлера поставленную начальную задачу на интервале $x \in [0, 1]$. Определяем правую часть уравнения (7) в виде функции `f1(x,y)`[4].

[4] В данном случае функция `f1` не зависит от второго аргумента, но в общем случае такая зависимость есть.

```
def f1(x, y):
    return 4 / (1 + x ** 2)
print(f1(1, 0))
```

```
2.0
```

3 Задаем начальное условие $y_0 = 0$ и $x_0 = 0$, а также верхнюю границу x_1 интервала, на котором мы будем искать решение[5].

[5] Нижняя граница — это x_0.

```
y0, x0, x1 = 0.0, 0.0, 1.0
print(y0, x0, x1)
```

```
0.0 0.0 1.0
```

4 Задаем число точек n для метода Эйлера и вычисляем шаг h.

```
n = 100000
h = (x1 - x0) / n
print(n, h)
```

```
100000 1e-05
```

5 Переходим собственно к методу Эйлера. Так как в данном случае нас интересует только конечное значение $y_n \approx \pi$, то нам не нужно запоминать все остальные промежуточные величины x_i и y_i. Поэтому мы можем ограничиться хранением только их текущих значений в переменных `x` и `y`.

```
y = y0
for i in range(n):
    x = x0 + i * h
    y = y + h * f1(x, y)
Pi = y
print(Pi, math.pi)
```

```
3.141602653573155
3.141592653589793
```

В первой строке приведенного фрагмента кода мы инициализируем переменную `y` начальным значением `y0`. Далее организуем цикл по всем точкам, на каждой его итерации вычисляем текущее значение `x` (по прямой формуле, чтобы не накапливать погрешность) и обновляем значение `y` по основной формуле метода Эйлера (3). После завершения цикла запоминаем последнее вычисленное значение в переменной `Pi` и выводим на печать полученную приближенную оценку числа π и «точное» значение этой константы из модуля `math`. Видно, что погрешность нашей оценки примерно равна 10^{-5}.

6 Теперь вычислим $\sqrt{\pi}$. Построим начальную задачу, решением которой будет функция $y(x) = \sqrt{x}$. Для этого заметим, что

$$y' = \frac{1}{2\sqrt{x}} \Rightarrow y' = \frac{1}{2y}. \tag{9}$$

Общим решением этого дифференциального уравнения[6] является функция $y(x) = C\sqrt{x}$. В данном случае константа C исключается с помощью начального условия $y(1) = 1$[7]. Решаем данную начальную задачу на интервале $x \in [1, \pi]$, в качестве верхней границы будем использовать наше приближенное значение `Pi`. В остальном схема применения метода Эйлера остается той же самой.

[6] С учетом положительности и x и y в контексте решаемой задачи.

[7] Подумайте, почему здесь не подходит нулевое начальное условие $y(0) = 0$.

```
def f2(x, y):
    return 1 / (2 * y)
y0, x0, x1 = 1.0, 1.0, Pi
h = (x1 - x0) / n
y = y0
for i in range(n):
    x = x0 + i * h
    y += h * f2(x, y)
sqrtPi = y
print(sqrtPi, math.sqrt(math.pi))
```

```
1.7724584007845334
1.7724538509055159
```

Запоминаем вычисленное приближенное значение $\sqrt{\pi}$ в переменной `sqrtPi` и сравниваем его со значением, вычисленным с помощью модуля `math`. Видно, что погрешность вычисления не увеличилась.

7 Переходим к табулированию функции, представленной интегралом в (5). Для этого сначала построим таблицу значений подынтегральной функции

$$y(x) = e^{-x^2}, \tag{10}$$

которая называется гауссовой функцией. Вычисляем производную этой функции и упрощаем полученное выражение:

$$y' = -2xe^{-x^2} \Rightarrow y' = -2xy. \tag{11}$$

Получили дифференциальное уравнение с рациональной правой частью. Подбираем начальное условие, при котором решением данного уравнения будет искомая функция (10): $y(0) = 1$. Решаем поставленную начальную задачу методом Эйлера. В этот раз, в отличие от предыдущих случаев, нас интересуют значения функции во всех точках рассматриваемого интервала. Поэтому вычисленные приближенные значения будем запоминать в массиве `Y`. Кроме того, заранее вычислим с помощью команды `linspace` и сохраним в отдельном массиве `X` координаты узлов сетки $x_i = x_0 + ih$ на интервале $x \in [0, 3]$[8]. При использовании массивов вместо скаляров команда внутри цикла оказывается полностью идентична основной формуле (3). После завершения цикла построим график полученной табличной функции (рис. 3), представляющий собой половину гауссианы.

[8] Для значений $x > 3$ интересующая нас функция приблизительно равна нулю.

```
def f3(x, y):
    return - 2 * x * y
y0, x0, x1 = 1.0, 0.0, 3.0
h = (x1 - x0) / n
X = np.linspace(x0, x1, num = n + 1, endpoint = True)
Y = np.zeros(n + 1)
Y[0] = y0
for i in range(n):
    Y[i + 1] = Y[i] + h * f3(X[i], Y[i])
plt.plot(X, Y)
plt.show()
```

8 Выполним, наконец, табулирование искомой функции $z(x) = \operatorname{erf} x$. По формуле Лейбница для дифференцирования интеграла (5), зависящего от параметра, находим производную искомой функции:

$$z' = \frac{2}{\sqrt{\pi}} e^{-x^2} \Rightarrow z' = \frac{2}{\sqrt{\pi}} y, \tag{12}$$

где $y(x)$ — гауссова функция, представленная нами в табличном виде в форме массивов `X` и `Y`. Дифференциальное уравнение (12) является, таким образом, простейшим. Очевидным начальным условием будет нулевое условие $z(0) = 0$, т. к. при $x = 0$ пределы интеграла (5) совпадают и он обращается в ноль. Решаем методом Эйлера начальную задачу для $z(x)$, причем используем тот же массив `X` значений переменной x, а в качестве функции правой части используем массив `Y` значений функции $y(x)$, умноженный на дробь $2/\sqrt{\pi}$. Таблицу значений искомой функции запоминаем в массиве `Z`. Строим график полученной функции (рис. 4).

```
y0 = 0.0
Z = np.zeros(n + 1)
Z[0] = y0
for i in range(n):
    F = Y[i] * 2 / sqrtPi
    Z[i + 1] = Z[i] + h * F
plt.plot(X, Z)
plt.show()
```

9 Сравним визуально график табулированной нами функции ошибок с графиком той же функции, построенной с использованием модуля `math`. Второй график построим с помощью маркеров, чтобы графики не накладывались друг на друга (рис. 5).

```
plt.plot(X, Z, label = "tabulated")
XX = np.linspace(x0, x1, 20, endpoint = True)
E = np.array([math.erf(x) for x in XX])
```

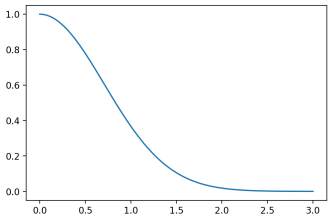


РИС. 3 График таблично заданной гауссовой функции

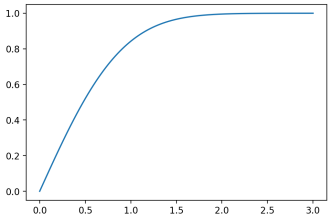


РИС. 4 График таблично заданной функции ошибок

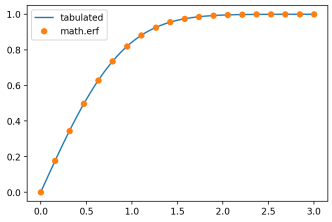


РИС. 5 Сравнение табулированной функции ошибок и функции `erf` модуля `math`

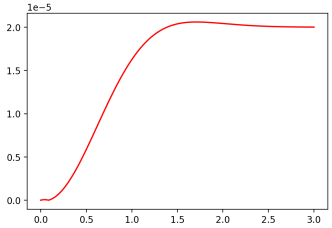


РИС. 6 График погрешности предложенного метода

[9] Во всех наших реализациях метода Эйлера использовалось одно и то же число точек n, однако так как длина интервала $x_1 - x_0$ была разной, то разными были и значения шага h.

[10] Предполагая, что x находится в том же диапазоне, что и элементы последовательности x_i.

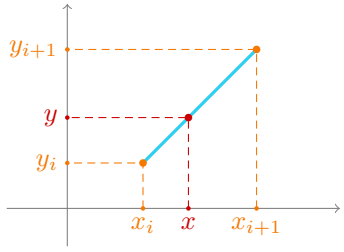


РИС. 7 Оценка функции $y(x)$ в точке $x \in [x_i, x_{i+1})$

```
plt.plot(XX, E, "o", label = "math.erf")
plt.legend()
plt.show()
```

10 Построим график модуля разности между табулированной функцией ошибок и ее реализацией в модуле `math` (рис. 6). Если считать, что функция `math.erf` является точной, то указанная разность может считаться погрешностью нашего метода табулирования. Из графика видно, что максимальная погрешность метода равна $2 \cdot 10^{-5}$ (см. масштабный множитель `1e-5` слева вверху графика), что примерно совпадает с порядком величины выбранного нами шага h[9].

```
ERF = np.array([math.erf(x) for x in X])
plt.plot(X, np.abs(Z - ERF), c = "red")
plt.show()
```

11 Если у нас есть подсчитанная с хорошей точностью табличная функция (x_i, y_i), $i = 0, 1, \ldots$, то мы можем реализовать на ее основе обычную функцию `Python`. По заданному аргументу x[10] сначала найдем пару элементов x_i и x_{i+1}, между которыми располагается x: $x_i \le x < x_{i+1}$. Если последовательность x_i сформирована с постоянным шагом h, то нужный нам индекс i вычисляется по простой формуле

$$i = \left\lfloor \frac{x - x_0}{h} \right\rfloor, \tag{13}$$

где внешние скобки означают операцию округления вниз. Если h мало, то на интервале $[x_i, x_{i+1}]$ функция $y(x)$ близка к линейной, поэтому ее значение в точке x можно оценить по простой пропорции (см. рис. 7)

$$y(x) = y_i + (y_{i+1} - y_i)\frac{x - x_i}{h}. \tag{14}$$

Применим этот прием для создания своей собственной функции `erf(x)`, работающей в определенном нами диапазоне `(x0, x1)`. Табличные значения представлены вычисленными ранее массивами `X` и `Z`.

```
def erf(x):
    i = math.floor((x - x0) / h)
    z1, z2 = Z[i], Z[i + 1]
    return z1 + (z2 - z1) * (x - X[i]) / h
```

12 Для примера вычислим значение $\mathrm{erf}(\pi/4)$ в точке, которой точно нет в массиве `X`, и сравним его с аналогичным значением, вычисленным с помощью функции `erf` из модуля `math`.

```
x = math.pi / 4
print(erf(x), math.erf(x))
```

```
0.7333236883334002
0.7333114255659122
```

Два найденных значения различаются в пятом знаке, что соответствует полученной выше оценке точности нашего метода.

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Перепишите рассмотренную выше функцию `erf`, чтобы она работала для всех $x \in \mathbb{R}$. Для отрицательных x используйте свойство нечетности, для $x > x_1$ — свойство, что $\operatorname{erf} x \approx 1$. Оцените величину x_1, так чтобы погрешность функции `erf(x)` при $x > x_1$ не превышала ее погрешности на интервале $x \in [0, x_1]$.

2 Постройте графики погрешности ε для задачи табулирования функции ошибок методом Эйлера для разных значений параметра $n = 100, 1000, \ldots$. Если эти графики построить в логарифмическом масштабе по оси Oy, используя команду `plt.yscale("log")` (см. рис. 8), то можно заметить, что погрешность ε уменьшается в 10 раз при увеличении n в 10 раз, т. е. при уменьшении шага h в 10 раз, другими словами:

$$\varepsilon = O(h). \tag{15}$$

Говорят, что некоторый метод имеет *k-й порядок точности* при решении дифференциального уравнения, если его погрешность ε на рассматриваемом интервале пропорциональна величине h^k. Следовательно, метод Эйлера имеет первый порядок точности[11].

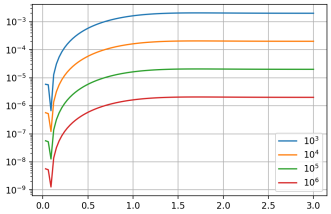


РИС. 8 Погрешность метода Эйлера для разных значений параметра n

3 Источнник появления погрешности в методе Эйлера можно увидеть на рис. 1. Если вторая производная искомой функции не равна нулю, то интересующая нас интегральная кривая в точке x_{i+1} уйдет либо выше, либо ниже касательной в точке x_i. То есть на самом деле нам нужна не касательная, а секущая. Можно частично скомпенсировать возникающую ошибку с помощью так называемого метода *предиктор—корректор*. Сначала выполняется предсказание нового значения y_{i+1} с помощью формулы Эйлера (3):

$$\tilde{y} = y_i + h \cdot k_1, \text{ где } k_1 = f(x_i, y_i).$$

[11] Этот факт можно строго доказать методами математического анализа.

Затем вычисляется угловой коэффициент в предсказанной точке $k_2 = f(x_{i+1}, \tilde{y})$. Наконец, в качестве углового коэффициента нужной нам секущей берется среднее арифметическое коэффициентов k_1 и k_2. Соответственно, в окончательном виде данный метод записывается следующим образом:

$$\begin{aligned} k_1 &= f(x_i, y_i), \\ k_2 &= f(x_i + h, y_i + h \cdot k_1), \\ y_{i+1} &= y_i + h \cdot (k_1 + k_2)/2. \end{aligned} \tag{16}$$

С помощью разложения в ряд Тэйлора можно строго показать, что такой метод имеет точность второго порядка. Решите рассмотренную в главе задачу с помощью данного метода. Сравните точность найденного решения с точностью метода Эйлера.

Брук Тейлор

Карл Давид Тольме Рунге

Мартин Вильгельм Кутта

4 Стандартом де-факто приближенного решения дифференциальных уравнений является метод Рунге—Кутты четвертого порядка, который вычисляет новое значение искомой функции с помощью серии из нескольких предсказаний и коррекций:

$$\begin{aligned} k_1 &= f(x_i, y_i), \\ k_2 &= f(x_i + h/2, y_i + h/2 \cdot k_1), \\ k_3 &= f(x_i + h/2, y_i + h/2 \cdot k_2), \\ k_4 &= f(x_i + h, y_i + h \cdot k_3), \\ y_{i+1} &= y_i + h \cdot (k_1 + 2k_2 + 2k_3 + k_4)/6. \end{aligned} \tag{17}$$

Простой геометрической интерпретации этот метод не имеет, формулы (17) были найдены алгебраическим способом из условия, чтобы данный метод имел четвертый порядок точности[12].

5 Оформите метод Эйлера, метод предиктор—корректор (см. упр. 3) и метод Рунге—Кутты 4-го порядка (см. упр. 4) в виде функций `Python`, принимающих на вход: правую часть дифференциального уравнения $f(x, y)$; диапазон изменения переменной (x_0, x_1); начальное значение y_0; число узлов сетки n. Каждая функция должна возвращать два массива `NumPy`: массив координат узлов сетки и массив соответствующих приближенных значений решения дифференциального уравнения.

6 Решите с помощью какого-нибудь приближенного метода следующие начальные задачи:

1) $y' = y(2 - y)$, $y(0) = 1$;
2) $y' = y/x - 1$, $y(1) = 1$;
3) $y' = \sin 2x - y$, $y(0) = 0$;
4) $y' = 2y^2(x + 1)$, $y(-1) = -1$.

[12] Метод Эйлера иногда называют методом Рунге—Кутты первого порядка точности, метод предиктор—корректор — методом Рунге—Кутты второго порядка точности.

Сравните приближенное решение с аналитическим, найденным с помощью библиотеки SymPy.

7 Поставьте начальные задачи для вычисления следующих математических констант:

$$\sqrt{2},\ \sqrt[3]{2},\ \sqrt{e},\ \frac{1}{e},\ e^{\pi}.$$

Выполните приближенные вычисления соответствующих констант с помощью какого-нибудь из рассмотренных выше приближенных методов. Постройте графики зависимости погрешности оценки констант от величины шага h.

8 Поставьте начальную задачу, решением которой является заданная функция, и выполните табулирование этой функции в указанном диапазоне изменения ее аргумента с помощью одного из приближенных методов решения дифференциальных уравнений:

1) e^{-x}, $x \in [0, 10]$;
2) $\ln(1 + x)$, $x \in [0, 5]$;
3) $\sqrt{1 - x^2}$, $x \in [0, 1]$;
4) 2^x, $x \in [0, 1]$.

Иоганн Генрих Ламберт

9 Придумайте схему табулирования функции $y^{-1}(x)$, обратной к заданной своей таблицей функции $y(x)$. Используя эту схему, постройте таблицу значений функции Ламберта $W(x)$, обратной к функции $y(x) = xe^x$.

10 Для табулирования функций sin и cos можно использовать начальную задачу для системы из двух обыкновенных дифференциальных уравнений[13]:

$$y' = z,\ z' = -y,\ y(0) = 0,\ z(0) = 1. \tag{18}$$

Метод Эйлера обобщается для приближенного решения системы (18) тривиальным образом:

$$\begin{aligned} &y_0 = 0,\ z_0 = 1,\\ &y_{i+1} = y_i + h \cdot z_i,\ z_{i+1} = z_i - h \cdot y_i\\ &i = 0, 1, \ldots \end{aligned} \tag{19}$$

Вычислите с помощью такой начальной задачи число $\sin 1$.

11 Выполните табулирование обратных тригонометрических функций $\arcsin x$ и $\arccos x$.

12 Придумайте способ приближенного решения дифференциального уравнения с заданным начальным условием справа налево, т. е. в сторону *уменьшения* аргумента. Используйте этот способ, например, для составления таблицы функции e^x (как решения начальной задачи $y' = y$, $y(0) = 1$) в отрицательном диапазоне значений переменной x.

[13] Единственное решение этой системы $y = \sin x$, $z = \cos x$ может быть найдено аналитически.

13 Рассмотрим начальную задачу

$$y' = 1 + y^2,\ y(0) = 0. \tag{20}$$

Хотя правая часть дифференциального уравнения не имеет никаких особенностей (не обращается в ноль, определена везде, т. е. не имеет никаких разрывов), решение данной задачи $y(x) = \operatorname{tg} x$ имеет разрыв, например, при $x = \pi/2$. Попробуйте решить эту задачу с помощью метода Эйлера на интервале $[0, 2]$. Что происходит с приближенным решением при переходе через точку разрыва $x = \pi/2$?

14 Сравните время работы реализованной нами функции `erf` с использованием предварительно составленной таблицы ее значений со временем работы аналогичной функции в модуле `math`.

ГЛАВА 8

Ортогональные траектории

Ортогональные семейства кривых • Два однопараметических семейства кривых называются взаимно ортогональными, если каждая кривая первого семейства пересекает каждую кривую второго семейства под прямым углом[1].

Ортогональные траектории широко применяются в разных разделах математики и физики. В прикладной математике такие семейства используются, например, в качестве криволинейных ортогональных систем координат (рис. 1). В физике взаимно ортогональными являются, например, семейства силовых линий заданного электростатического поля и соответствующих этому полю эквипотенциальных линий, на каждой из которых все точки имеют одинаковое значение электростатического потенциала (рис. 2).

Обобщением эквипотенциальных линий является понятие *изолинии*, в каждой точке которой измеряемая величина сохраняет одинаковое значение. Изолинии широко применяются в прикладных науках при визуализации различных двумерных величин, например высот в топографии (рис. 3), давлений в метеорологии (*изобары*, рис. 4) и т. п. Важным свойством изолиний является то, что ортогональные к ним кривые всегда показывают направление наибольшего возрастания (убывания) соответствующей величины.

Задача • Построение по заданному семейству кривых ортогонального к нему семейства является классической геометрической задачей в теории дифференциальных уравнений. Схема решения этой задачи следующая: 1) по заданному семейству кривых строим дифференциальное уравнение, общим решением кото-

[1] Под углом между кривыми понимается угол между их касательными в точке пересечения.

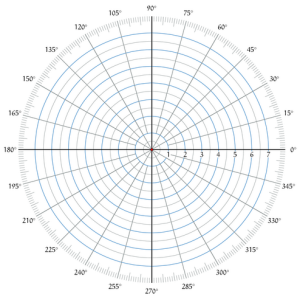


РИС. 1 Полярная система координат

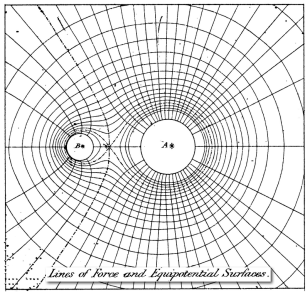


РИС. 2 Силовые и эквипотенциальные линии (*рисунок Джеймса Максвелла*)

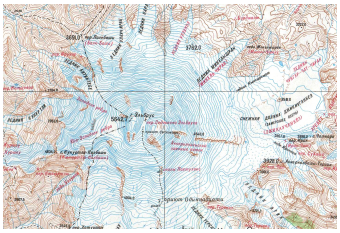


РИС. 3 Линии одинаковых высот, топографическая карта Эльбруса

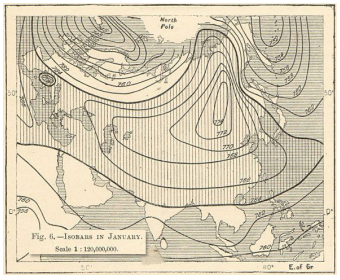


РИС. 4 Изобары — линии одинакового давления

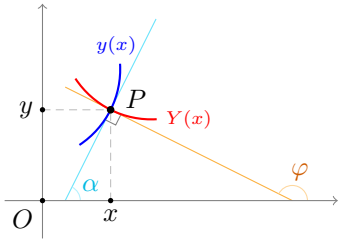


РИС. 5 Связь между углами наклона касательных к двум взаимно ортогональным кривым $y(x)$ и $Y(x)$

рого является это семейство; 2) преобразуем полученное уравнение в уравнение, соответствующее ортогональному семейству кривых; 3) решаем построенное уравнение, найденное его общее решение описывает искомое семейство ортогональных траекторий.

Если исходное семейство кривых задано «явным» соотношением[2]

$$f(x,y)=C, \tag{1}$$

то для построения искомого дифференциального уравнения достаточно продифференцировать по переменной x обе части этого соотношения:

$$\frac{\partial f}{\partial x}+\frac{\partial f}{\partial y}\cdot\frac{dy}{dx}=0. \tag{2}$$

Обозначим построенное таким образом дифференциальное уравнение как

$$\Phi(x,y,y')=0. \tag{3}$$

Рассмотрим (рис. 5) теперь произвольную точку P с координатами (x,y) и проходящую через нее интегральную кривую уравнения (3), являющуюся представителем исходного семейства кривых (1). Пусть искомая ортогональная кривая, проходящая через точку P, описывается функцией $Y(x)$. Согласно определению угла между двумя кривыми, касательные к кривым $y(x)$ и $Y(x)$ в точке P перпендикулярны. Следовательно, для углов α и φ наклона этих касательных к оси Ox справедливо следующее соотношение:

$$\varphi=\frac{\pi}{2}+\alpha \Rightarrow \operatorname{tg}\varphi=-\operatorname{ctg}\alpha=-\frac{1}{\operatorname{tg}\alpha}. \tag{4}$$

Так как $\operatorname{tg}\alpha = y'(x)$, а $\operatorname{tg}\varphi = Y'(x)$, то в силу (4) производные функций $y(x)$ и $Y(x)$ оказываются связанными в точке x равенством

$$y'(x)=-\frac{1}{Y'(x)}. \tag{5}$$

Кроме того, $y(x)=Y(x)$ в данной конкретной точке x. Поэтому если функция $y(x)$ удовлетворяет дифференциальному уравнению

$$\Phi(x,y,y')=0,$$

[2] См. упражнение 3 к главе для общего «неявного» случая

$$F(x,y,C)=0.$$

то искомая функция $Y(x)$ должна удовлетворять уравнению

$$\Phi\left(x, Y, -\frac{1}{Y'}\right) = 0. \tag{6}$$

Заменив теперь в (6) Y на y (это всего лишь имя функции), получим следующее правило: дифференциальное уравнение, описывающее семейство ортогональных кривых к семейству кривых, заданному дифференциальным уравнением (3), получается заменой в этом уравнении производной y' на $-1/y'$:

$$\Phi(x, y, y') = 0 \rightsquigarrow \Phi\left(x, y, -\frac{1}{y'}\right) = 0. \tag{7}$$

Общее решение построенного таким способом дифференциального уравнения, содержащее произвольный параметр C, и будет определять искомое однопараметрическое семейство ортогональных траекторий.

Модель • В качестве примера рассмотрим задачу построения ортогональных траекторий для семейства степенных функций $y = C_1 x^n$ при фиксированной степени n.

1 Подключаем библиотеки `SymPy` и `NumPy` и функцию `display`.

2 Определяем символы x, C_1, n и функцию $y(x)$. Задаем исходное семейство `fam` степенных функций.

```
x, C1, n = symbols("x C1 n")
y = Function("y")(x)
fam = Eq(y, C1 * x ** n)
display(fam)
```

$$y(x) = C_1 x^n$$

3 Визуализацию нашей модели рассмотрим в частном случае $n = -1$.

```
hyp = fam.subs(n, -1)
display(hyp)
```

$$y(x) = \frac{C_1}{x}$$

4 Построим семейство кривых `hyp`. Так как в общем случае подобные семейства определяются в неявной

форме, то для визуализации семейства `hyp` будем использовать команду `plot_implicit` для построения графиков неявных функций. Перед построением каждого графика нужно подставить в `hyp` вместо `C1` соответствующее числовое значение и заменить символ функции `y` на предварительно определенный вспомогательный символ `Y`.

```
Y = symbols("y")
p1 = plot(xlim = (-5, 5), ylim = (-5, 5),
          size = (5, 5), show = False)
for c in np.arange(-10, 10, 0.5):
    f = hyp.subs({C1: c, y: Y})
    p = plot_implicit(f, (x, -5, 5), (Y, -5, 5),
                      show = False)
    p1.extend(p)
p1.show()
```

Результат выполнения данного фрагмента кода показан на рис. 6.

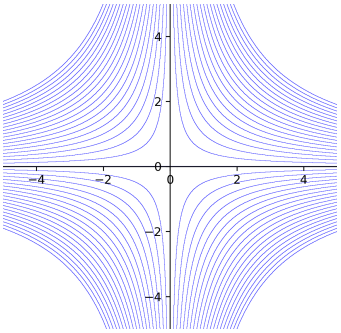


РИС. 6 Исходное семейство кривых для случая $n = -1$

5 Приведем исходное уравнение `fam` к виду 1, разрешив данное уравнение относительно символа `C1`.

```
csol = solveset(fam, C1).args[0]
display(Eq(csol, C1))
```

$$x^{-n}y(x) = C_1$$

6 Составляем дифференциальное уравнение, описывающее исходное семейство кривых. Для этого дифференцируем выражение `csol` по `x` и приравниваем вычисленную производную к нулю (производная константы в правой части в последнем соотношении).

```
ode = Eq(diff(csol, x), 0)
display(ode)
```

$$-\frac{nx^{-n}y(x)}{x} + x^{-n}\frac{d}{dx}y(x) = 0$$

7 Решим полученное дифференциальное уравнение, чтобы убедиться, что оно в самом деле описывает исходное семейство кривых. Для решения используем подсказку `separable`[3].

```
dsol = dsolve(ode, y, hint = "separable")
display(dsol)
```

[3] Попробуйте решить данное уравнение без подсказок!

$$y(x) = C_1 e^{n \log(x)}$$

8 После упрощения полученного соотношения с помощью команды `logcombine`, как и предполагалось, получаем исходное уравнение `fam`.

```
dsol = logcombine(dsol, force = True)
display(dsol)
```

$$y(x) = C_1 x^n$$

9 По формуле (7) преобразуем уравнение `ode` к уравнению `ode2`, описывающему искомое ортогональное семейство кривых.

```
ode2 = ode.subs(diff(y, x), -1 / diff(y, x))
display(ode2)
```

$$-\frac{n x^{-n} y(x)}{x} - \frac{x^{-n}}{\frac{d}{dx} y(x)} = 0$$

10 Упростим полученное уравнение. Для этого сначала разрешим его относительно производной.

```
sol = solveset(ode2, diff(y, x))
display(sol)
```

$$\left\{-\frac{x}{n y(x)}\right\} \setminus \{0\}$$

11 Извлечем из полученного выражения собственно решение — выражение в фигурных скобках слева[4].

```
rhs2 = sol.args[0].args[0]
display(rhs2)
```

$$-\frac{x}{n y(x)}$$

12 Переопределяем уравнение `ode2`, приравнивая к производной `diff(y,x)` найденное выражение `rhs2`.

```
ode2 = Eq(diff(y, x), rhs2)
display(ode2)
```

[4] Выражение справа задает область допустимых значений для данного уравнения, т. е. в нашем случае $y' \neq 0$.

$$\frac{d}{dx}y(x) = -\frac{x}{ny(x)}$$

13 Решаем дифференциальное уравнение `ode2`. Чтобы получить ответ в виде одного неявного выражения, используем для решения комбинацию опций `hint` и `simplify`.

```
dsol2 = dsolve(ode2, y, hint = "separable",
               simplify = False)
display(dsol2)
```

$$\frac{y^2(x)}{2} = C_1 - \frac{x^2}{2n}$$

14 По описанной выше схеме строим график найденного семейства кривых `dsol2` для частного случая $n = -1$ (см. рис. 7).

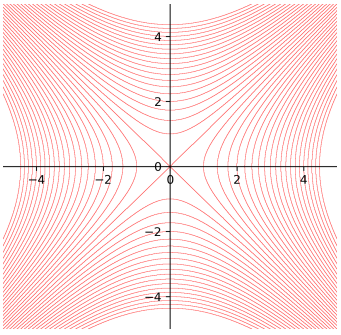


РИС. 7 Семейство искомых ортогональных траекторий для случая $n = -1$

```
p2 = plot(xlim = (-5, 5), ylim = (-5, 5),
          size = (5, 5), show = False)
for c in np.arange(-10, 10, 0.5):
    f = dsol2.subs({C1: c, y: Y, n: -1})
    p = plot_implicit(f, (x, -5, 5), (Y, -5, 5),
                      line_color = "red",
                      show = False)
    p2.extend(p)
p2.show()
```

15 Последним действием объединяем оба семейства кривых на одном графике.

```
p3 = plot(xlim = (-5, 5), ylim = (-5, 5),
          size = (5, 5), show = False)
p3.extend(p1)
p3.extend(p2)
p3.show()
```

Окончательный результат показан на рис. 8.

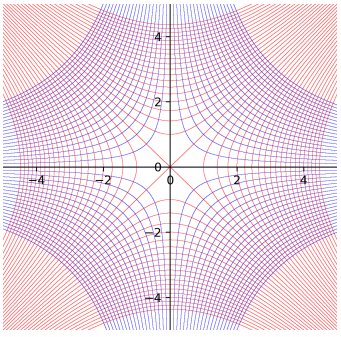


РИС. 8 Два семейства взаимно ортогональных траекторий для случая $n = -1$

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Постройте графики семейств взаимно ортогональных траекторий в рассмотренной в главе модели для следующих частных случаев:

1) $n = 1$;
2) $n = 2$;
3) $n = 1/3$;
4) $n = -2$.

2 Постройте с помощью `SymPy` дифференциальные уравнения, описывающие заданные семейства кривых. Для разрешения уравнения относительно константы используйте команду `solve`, если команда `solveset` не работает.

1) $y = C \cdot \ln x$;
2) $y = e^{Cx}$;
3) $y = \sin(x - C)$;
4) $y^2 + Cx^2 = C^2$.

3 В более общем случае, когда семейство кривых задается в виде неявного относительно параметра C соотношения $F(x, y, C) = 0$, это соотношение надо также продифференцировать по x, после чего из полученной системы

$$\begin{aligned} &F(x, y, C) = 0, \\ &\frac{\partial F}{\partial x} + \frac{\partial F}{\partial y} \cdot \frac{dy}{dx} = 0 \end{aligned} \tag{8}$$

алгебраически исключить C. Найденное таким образом соотношение будет включать в себя x, y и y', т. е. будет искомым дифференциальным уравнением. Постройте по такой схеме дифференциальные уравнения, описывающие следующие семейства кривых:

1) $y = Cx + C^2$;
2) $y = \log(x - C) + C$;
3) $Cy = e^{Cx}$;
4) $\sin(Cy) + Cx = 0$.

4 Постройте ортогональные траектории для следующих семейств кривых:

1) $x^2 + (y - C)^2 = C^2$;
2) $x - y = Ce^x$;
3) $\cos y = C \sin x$;
4) $\sin y = Ce^{x^2}$.

Если в решении дифференциального уравнения появляются логарифмы, то перед построением графиков этого решения замените каждое выражение под логарифмом на его модуль с помощью метода `replace`, как это сделано в следующем примере:

```
e1 = log(x - 1) + log(x + 1)
e2 = e1.replace(log, lambda e: log(abs(e)))
display(e1, e2)
```

$$\log(x - 1) + \log(x + 1)$$
$$\log(|x - 1|) + \log(|x + 1|)$$

5 Для заданной функции $f(x)$ постройте[5] семейство ортогональных траекторий для семейства кривых, заданных соотношением $y = f(x) + C$ (см. рис. 9):

1) $f(x) = x^3$;
2) $f(x) = \cos x$;
3) $f(x) = 1/(1 + x^2)$;
4) $f(x) = 1/\sin x$.

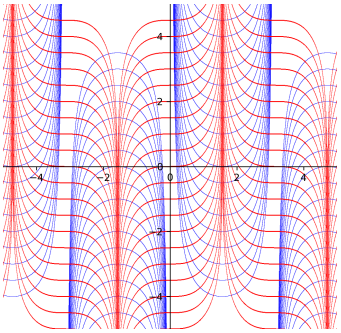


РИС. 9 Ортогональные траектории для семейства кривых $y = 1/\sin x + C$

[5] В данном случае все дифференциальные уравнения решаются хорошо в явном виде, поэтому для построения графиков обоих семейств можно использовать значительно быстрее работающую команду `plot`.

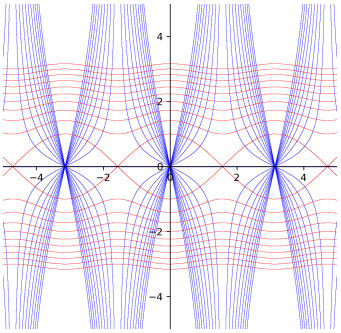


РИС. 10 Ортогональные траектории для семейства кривых $y = C\,\mathrm{tg}\,x$

6 Для заданной функции $f(x)$ постройте семейство ортогональных траекторий семейства кривых, заданного соотношением $y = Cf(x)$ (см. рис. 10):

1) $f(x) = \ln x$;
2) $f(x) = 1 + 1/x$;
3) $f(x) = \sin x$;
4) $f(x) = \mathrm{tg}\,x$.

7 Для заданной функции $f(x)$ постройте семейство ортогональных траекторий семейства кривых, заданного соотношением $y = f(x - C)$:

1) $f(x) = \sin x$;
2) $f(x) = 1/x$;
3) $f(x) = \log x$;
4) $f(x) = \arccos x$.

8 Для заданной[6] функции $f(x)$ постройте семейство ортогональных траекторий семейства кривых, заданного соотношением $y = f(Cx)$:

1) $f(x) = 1/(1 + x)$;
2) $f(x) = \sqrt{x + 1}$;
3) $f(x) = \mathrm{arctg}\,x$;
4) $f(x) = \mathrm{arch}\,x$.

9 Чтобы получить ортогональные траектории семейства кривых, заданных дифференциальным уравнением

$$F(\varphi, r, r') = 0 \tag{9}$$

в полярной системе координат, нужно заменить в этом уравнении производную r' на выражение $-r^2/r'$:

$$F(\varphi, r, r') = 0 \rightsquigarrow F(\varphi, r, -r^2/r') = 0. \tag{10}$$

Это обусловлено тем, что в полярных координатах тангенс угла α наклона касательной к кривой $r = r(\varphi)$ к радиусу в данной точке равен r/r' (см. рис. 11). При переходе к ортогональной траектории угол α меняется на $\alpha + \pi/2$, поэтому $\mathrm{tg}\,\alpha$ меняется на $-1/\mathrm{tg}\,\alpha$, т. е. $r/r' \to -r'/r$, что и дает указанную выше замену. Используя это правило, постройте ортогональные траектории следующих семейств кривых в полярной системе координат ($\varphi > 0$):

1) $r = C\sin^2\varphi$;
2) $r = C\cos\varphi$;
3) $r = C\varphi^2$;
4) $r = C\,\mathrm{tg}\,\varphi$.

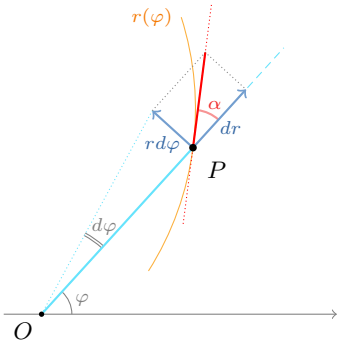


РИС. 11 Геометрический смысл производной $r'(\varphi)$ в полярной системе координат

10 Кривая, которая пересекает заданное однопараметрическое семейство кривых под одним и тем же углом, называется *изогональной траекторией* данного семейства кривых. Метод построения семейства изогональных траекторий аналогичен построению семейств ортогональных траекторий. Сначала составляем дифференциальное уравнение, общим решением которого является исходное семейство кривых[7]:

$$y' = f(x, y). \tag{11}$$

[6] Функция $\mathrm{arctg}\,x$ в SymPy реализована командой `atan(x)`, обратный гиперболический косинус (ареакосинус) $\mathrm{arch}\,x$ — командой `acosh(x)`.

[7] В данном случае проще рассмотреть дифференциальное уравнение с явно выраженной производной.

Пусть угол, под которым изогональные траектории должны пересекать наше семейство, равен γ, по определению, это угол между соответствующими касательными. Выберем на одной из исходных кривых произвольную точку P и рассмотрим $\triangle ABP$ (рис. 12), образованный осью Ox и двумя касательными: PA — к исходной кривой, угол ее наклона равен α, PB — к искомой изогональной кривой, угол ее наклона равен β. Так как β — внешний угол треугольника, то $\beta = \alpha + \gamma$. Следовательно, $\operatorname{tg}\beta = \operatorname{tg}(\alpha+\gamma)$. Применяем формулу тангенса суммы:

$$\operatorname{tg}\beta = \frac{\operatorname{tg}\alpha + \operatorname{tg}\gamma}{1 - \operatorname{tg}\alpha\operatorname{tg}\gamma}. \tag{12}$$

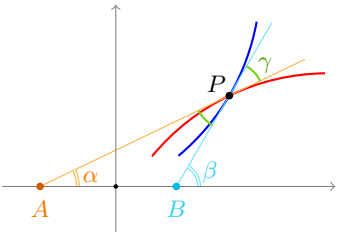


РИС. 12 Соотношение между углами α и β наклона двух кривых к оси Ox и углом γ между этими кривыми

Тангенс угла β равен производной y' *искомой* функции, описывающей изогональную траекторию, тангенс угла α равен производной *исходной* функции, которая согласно уравнению (11) равна $f(x,y)$. Подставляя эти выражения в (12), получаем следующее соотношение:

$$y' = \frac{f(x,y) + k}{1 - k\cdot f(x,y)}, \tag{13}$$

где $k = \operatorname{tg}\gamma$. Так как соотношение (13) должно выполняться для всех точек всех кривых исходного семейства, то оно и является искомым дифференциальным уравнением для изогональных траекторий заданного семейства кривых.

11 Для следующих семейств кривых постройте их изогональные траектории с заданным значением параметра k:

1) $y = x + C$, $k = 0.5$;
2) $y = x^2 + C$, $k = 2$;
3) $y^2 = 4Cx$, $k = 1$;
4) $x^2 + 2xy - y^2 = C^2$, $k = 1$.

12 Для построения семейств кривых, заданных неявным соотношением $F(x,y) = C$, в некоторых случаях оказывается намного быстрее и удобнее использовать команду `contour` из библиотеки `Matplotlib`, предназначенную для построения линий уровня заданной функции двух переменных, а не команду `plot_implicit` из библиотеки `SymPy`. Например, пусть имеется выражение[8] `SymPy` (см. код ниже, строка 2)

$$F(x,y) = \sin(x+y)\cos(y-x). \tag{14}$$

Необходимо построить семейство кривых вида

$$F(x,y) = C \tag{15}$$

для некоторого заданного набора значений параметра C. Преобразуем выражение `F` в функцию `f` с помощью команды `lambdify` (строка 3). Создаем сетку в области построения графиков командой `meshgrid` библиотеки `NumPy`

[8] Если в выражение входит символ функции, то его надо предварительно заменить на символ переменной, так же как это мы делали при построении графиков неявной функции.

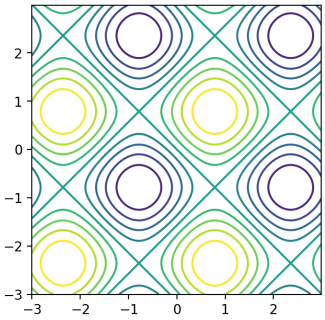


РИС. 13 Линии уровня, построенные с помощью библиотеки `Matplotlib`

(строки 4–6). Строим линии уровня функции `f`, используя команду `contour`. Первые два ее аргумента задают сетку, третий аргумент — сеточную функцию (по сути, матрицу значений рассматриваемого выражения в узлах сетки), четвертый аргумент определяет список уровней (значений параметра C), которые мы хотим визуализировать.

```
1 x, y = symbols("x y")
2 F = sin(x + y) * cos(y - x)
3 f = lambdify([x, y], F)
4 xrange = np.arange(-3.0, 3.0, 0.01)
5 yrange = np.arange(-3.0, 3.0, 0.01)
6 X, Y = np.meshgrid(xrange, yrange)
7 plt.contour(X, Y, f(X, Y), np.arange(-1, 1, 0.2))
8 plt.gca().set_aspect('equal')
9 plt.show()
```

Результат выполнения этого фрагмента кода показан на рис. 13. Заметим, что команда `contour`, в отличие от команды `plot_implicit`, имеет много опций, управляющих внешним видом рисуемых ею кривых.

ГЛАВА 9
Математическая вышивка

Задача • Рассмотрим следующую геометрическую задачу (см. рис. 1): найти кривую, касательная к которой в произвольной ее точке P пересекает координатные оси в таких точках $A(a,0)$ и $B(0,b)$, что сумма координат a и b является заданной постоянной величиной k:

$$a+b=k. \tag{1}$$

Уравнение касательной к кривой $y=y(x)$ в точке $x=x_0$, как известно, выглядит следующим образом:

$$y=y'(x_0)\cdot(x-x_0)+y(x_0). \tag{2}$$

Подставляя в эту формулу $y=0$ и $x=0$, найдем координаты пересечения касательной с осями Ox и Oy соответственно:

$$a=x_0-\frac{y(x_0)}{y'(x_0)},\ b=y(x_0)-x_0\cdot y'(x_0). \tag{3}$$

Заменяем в последних двух формулах x_0, $y(x_0)$ и $y'(x_0)$ на x, y и y' соответственно и подставляем полученные выражения для a и b в условие задачи (1):

$$\underbrace{x-y/y'}_{a}+\underbrace{y-y'\cdot x}_{b}=k. \tag{4}$$

Выражаем в последнем равенстве y через x и y' и после очевидных алгебраических преобразований получаем так называемое дифференциальное уравнение Клеро:

$$y=xy'+\frac{ky'}{y'-1}. \tag{5}$$

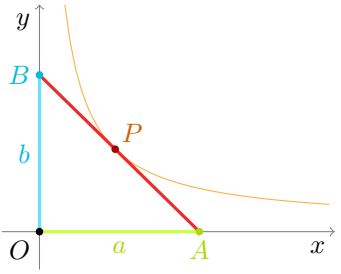


РИС. 1 Задача о постоянной сумме координат пересечения касательной с осями Ox и Oy

Уравнение Клеро • Уравнением Клеро называется нелинейное дифференциальное уравнение вида

$$y = xy' + \varphi(y'). \tag{6}$$

Уравнение Клеро, как и некоторые другие типы уравнений первого порядка, неразрешенные относительно производной, решается методом введения параметра. Обозначим $y' = p$, тогда

Алекси Клод Клеро

$$y = px + \varphi(p). \tag{7}$$

Продифференцируем последнее соотношение, учитывая, что p — это функция от x:

$$y' = p'x + p + \varphi'(p). \tag{8}$$

Заменяя слева y' на p, получаем после упрощения, что

$$p' \cdot (x + \varphi'(p)) = 0. \tag{9}$$

Это значит, что либо $p' = 0$, либо $x = \varphi'(p)$. В первом случае $p = C$, подставляем это выражение в (7), это дает нам общее решение уравнения 6:

$$y = Cx + \varphi(C). \tag{10}$$

То есть интегральными кривыми, соответствующими общему решению уравнения Клеро, является семейство прямых линий.

Во втором случае $x = -\varphi'(p)$, подставляем это выражение в (7) и находим еще одно решение уравнения Клеро уже в параметрической форме:

$$\begin{aligned} x &= -\varphi'(p), \\ y &= -p \cdot \varphi'(p) + \varphi(p). \end{aligned} \tag{11}$$

Последнее решение называется *особым* (или *сингулярным*). Особое решение не получается из общего решения того же дифференциального уравнения ни при каких значениях константы C. Кроме того, особое решение само не содержит никаких констант, т. е. задает только одну интегральную кривую.

Таким образом, полученное нами выше дифференциальное уравнение (5) является уравнением Клеро, в котором $\varphi(p) = kp/(p-1)$.

Модель • Решим поставленную задачу с помощью библиотеки SymPy. Чтобы иметь возможность обобщить применяемый подход к решению других аналогичных задач, начнем процесс построения модели с вывода дифференциального уравнения.

1 Подключаем библиотеки SymPy и NumPy и команду display.

2 Определяем символ x и функцию $y(x)$, которые мы будем использовать для построения дифференциального уравнения.

3 Чтобы корректно задать уравнение касательной к кривой $y(x)$ в точке x, нам потребуется еще одна пара символов X и Y, с помощью которых перепишем уравнение (2):

$$Y = y'(x)(X - x) + y(x). \tag{12}$$

Вводим новые символы и записываем уравнение касательной в SymPy.

```
X, Y = symbols("X Y")
tangent = Eq(Y, diff(y, x) * (X - x) + y)
display(tangent)
```

$$Y = (X - x)\,\frac{d}{dx}y(x) + y(x)$$

4 Находим пересечение касательной с осью Ox, подставляя в уравнение касательной вместо Y ноль и решая полученное уравнение относительно X.

```
a = solveset(tangent.subs(Y, 0), X).args[0]
display(a)
```

$$x - \frac{y(x)}{\frac{d}{dx}y(x)}$$

5 Выполняем аналогичные действия[1] для нахождения точки пересечения с осью Oy.

```
b = solveset(tangent.subs(X, 0), Y).args[0]
display(b)
```

$$-x\frac{d}{dx}y(x) + y(x)$$

6 Вводим символ k и составляем дифференциальное уравнение, используя соотношение (1).

[1] В данном случае можно было обойтись без решения уравнения, так как уравнение касательной уже записано в форме, разрешенной относительно символа Y. Но для общности подхода применим команду solveset.

```
k = symbols("k")
ode = Eq(a + b, k)
display(ode)
```

$$-x\frac{d}{dx}y(x) + x + y(x) - \frac{y(x)}{\frac{d}{dx}y(x)} = k$$

7 Так как мы не собираемся решать полученное дифференциальное уравнение стандартным образом (с помощью команды `dsolve`), то сразу заменим в нем производную на параметр p, а уже потом перейдем к приведению этого уравнения к форме, стандартной для уравнения Клеро.

```
p = symbols("p")
ode2 = ode.subs(diff(y, x), p)
display(ode2)
```

$$-px + x + y(x) - \frac{y(x)}{p} = k$$

8 Разрешаем уравнение относительно $y(x)$.

```
rhs = solveset(ode2, y).args[0]
ode3 = Eq(y, rhs)
display(ode3)
```

$$y(x) = \frac{p\,(k + px - x)}{p - 1}$$

9 Проверяем, является ли это уравнение уравнением Клеро. Для этого правая часть `rhs` должна линейно зависеть от `x`, причем коэффициент перед `x` должен быть равным `p`. Чтобы в этом убедиться, вычислим производную по `x` правой части уравнения и сравним ее с `p`.

```
if simplify(diff(rhs, x)) == p:
    print("Clairaut's equation")
else:
    print("Not the Clairaut's equation")
```

```
Clairaut's equation
```

10 Подставим в правую часть `rhs` вместо символа `x` ноль, полученное таким образом выражение будет функцией $\varphi(p)$.

```
phi = rhs.subs(x, 0)
display(phi)
```

$$\frac{kp}{p-1}$$

11 Теперь мы можем записать наше уравнение в форме, стандартной для уравнения Клеро[2]. Заметим, что с точностью до порядка слагаемых полученное уравнение совпадает с выведенным нами аналитически уравнением (5).

[2] Этот шаг не является обязательным, т. к. дальше мы будем работать исключительно с выражением `phi`.

```
y_ = symbols("y'")
ode4 = Eq(y, p*x+phi).subs(p, y_)
display(ode4)
```

$$y(x) = \frac{ky'}{y'-1} + xy'$$

12 Выражение `ode3` (см. пункт 8), как следует из описанной выше схемы решения уравнений Клеро, является его общим решением, в котором роль константы интегрирования играет параметр `p`. Так как правая часть `rhs` этого решения является линейной по `x`, то интегральными кривыми общего решения будут прямые линии. Построим семейство этих линий. Для этого предварительно выразим параметр p через y_0 — координату точки пересечения соответствующей прямой с осью Oy. То есть фактически найдем решение, удовлетворяющее начальному условию $y(0) = y_0$. Сохраним найденное решение начальной задачи в переменной `dsol`.

```
y0 = symbols("y0")
sol = solveset(ode3.subs({x: 0, y: y0}), p)
display(sol)
p1 = sol.args[0].args[0]
dsol = ode3.subs(p, p1)
display(dsol)
```

$$\left\{\frac{y_0}{-k+y_0}\right\} \setminus \{1\}$$

$$y(x) = \frac{y_0\left(k + \frac{xy_0}{-k+y_0} - x\right)}{(-k+y_0)\left(\frac{y_0}{-k+y_0} - 1\right)}$$

13 Построим семейство кривых общего решения `dsol` для частного случая $k = 1$ (рис. 2).

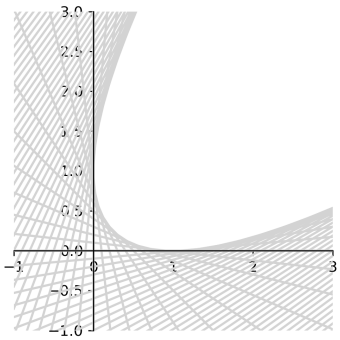


РИС. 2 Общее решение уравнения Клеро (5)

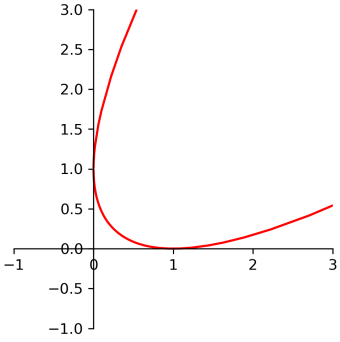


РИС. 3 Особое решение уравнения Клеро (5)

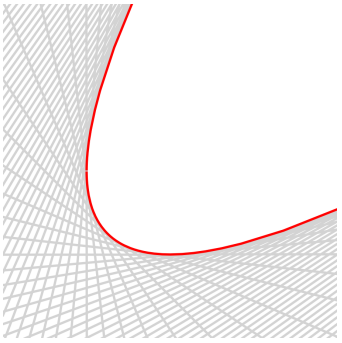


РИС. 4 Общее и особое решения уравнения (5)

```
pl1 = plot(xlim = (-1, 3), ylim = (-1, 3),
           aspect_ratio = (1, 1), show = False)
for y_0 in np.arange(-6, 6, 0.1):
    ds = dsol.subs({y0: y_0, k: 1})
    pl = plot(ds.rhs, (x, -1, 3),
              line_color = "lightgray", show = False)
    pl1.extend(pl)
pl1.show()
```

На построенном графике `pl1` хорошо видна криволинейная граница области, свободной от рассматриваемого семейства прямых, называемая *огибающей* этого семейства. Уравнение огибающей заданного семейства кривых часто является *особым* решением дифференциального уравнения, общим решением которого служит данное семейство кривых.

14 Найдем особое решение нашего уравнения Клеро согласно формулам (11).

```
solx = simplify(-diff(phi, p))
soly = simplify(p * solx + phi)
display(solx, soly)
```

$$\frac{k}{(p-1)^2}$$
$$\frac{kp^2}{(p-1)^2}$$

15 Строим график особого решения в параметрическом виде (рис. 3).

```
pl2 = plot(xlim = (-1, 3), ylim = (-1, 3),
           aspect_ratio = (1, 1), show = False)
sx, sy = solx.subs(k, 1), soly.subs(k, 1)
pl = plot_parametric((sx, sy), (p, -100, 100),
                     line_color = "red",
                     show = False)
pl2.extend(pl)
pl2.show()
```

16 Последним действием объединяем графики общего решения (семейство прямых серого цвета) и график особого решения (кривая красного цвета) в общий график. Для большей выразительности построим график без координатных осей, используя опцию `axis = False` команды `plot`.

```
pl3 = plot(xlim = (-1, 3), ylim = (-1, 3),
           aspect_ratio = (1, 1), axis = False,
           show = False)
pl3.extend(pl1)
```

```
pl3.extend(pl2)
pl3.show()
```

Окончательный результат решения поставленной задачи показан на рис. 4. Заметим, что описанные в пункте 13 действия (см. рис. 2), связанные с построением семейства прямых, соответствующих общему решению уравнения Клеро, легко выполнить руками, например, с помощью карандаша и линейки. Для этого надо соединить каждую точку на оси Ox с целочисленной координатой $x \in [1 \dots k]$ с точкой на оси Oy с координатой $y = k - x$ (рис. 5). Даже для небольших значений k огибающая семейства построенных таким образом отрезков выглядит достаточно «криволинейно». Такой прием широко применяется в специальной технике вышивания, называемой математической вышивкой (англ. *string art*), например для заполнения различных углов (рис. 6).

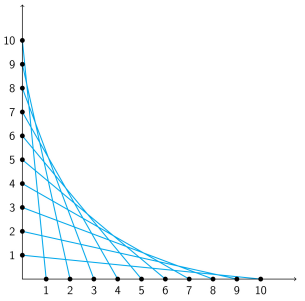


РИС. 5 Основной прием математического вышивания

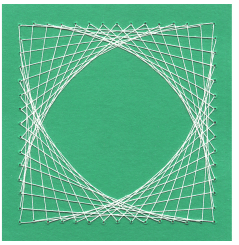


РИС. 6 Математическая вышивка

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Постройте на одном графике семейство кривых особого решения (11) уравнения (5) для разных значений параметра $k \in [-4, 4]$.

2 Проверьте с помощью SymPy, что любая прямая из найденного нами общего решения уравнения Клеро удовлетворяет условию задачи $a+b = k$, вычислив точки пересечения этой прямой с осями координат.

3 Рассмотрите решение более общей задачи, в которой касательная к кривой пересекается с двумя прямыми, описываемыми уравнениями $y = nx$ и $y = mx$. Сумма расстояний от точек пересечения до начала координат должна быть постоянной и равной k. Учтите, что расстояние a от точки с координатой $x = x_0$, лежащей на прямой $y = nx$, до начала координат равно

$$a = x_0\sqrt{1+n^2}. \tag{13}$$

Покажите, что и в такой конфигурации огибающей семейства прямых также будет парабола (рис. 7).

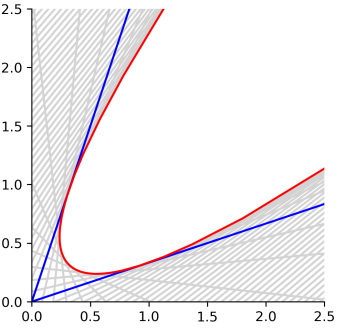


РИС. 7 Огибающая семейства отрезков, построенных на прямых $y = nx$ и $y = mx$

4 Постройте дифференциальное уравнение, общим решением которого является заданное семейство прямых линий:

1) $y = k^2x + 2k$;
2) $y = (k-1)x - k^3$;
3) $y = e^{-k}x + k^2$;
4) $y = \sin k \cdot x + \cos k$.

5 Найдите с помощью SymPy общее и особое решения следующих уравнений Клеро:

1) $y = xy' + y'^2$;
2) $y = xy' + \ln y'$;
3) $y = xy' - \sin y'$;
4) $y = xy' + (1 + y'^2)$.

6 Действия, которые мы выполняли в пунктах 9 и 10 для извлечения коэффициентов линейного выражения, можно выполнить проще, используя команду `linear_coeffs`, определенную в модуле `sympy.solvers.solveset`. Первым аргументом этой команде передается выражение, которое мы хотим представить в виде линейной функции от символов, перечисленных последующими аргументами. Команда возвращает список коэффициентов этого линейного выражения комбинации[3] в том порядке, в котором перечислены символы. Последним элементом списка идет свободный коэффициент, не зависящий ни от одного из этих символов. Например, для выражения `rhs` эта команда выдаст следующий результат:

```
from sympy.solvers.solveset import linear_coeffs
coeffs = linear_coeffs(rhs, x)
display(coeffs)
```

```
[p, k*p/(p - 1)]
```

Видно, что коэффициент при `x` в данном случае равен `p` (что соответствует уравнению Клеро), а свободный коэффициент (т. е. функция $\varphi(p)$) равен $kp/(p-1)$.

7 Напишите две функции для работы с дифференциальными уравнениями Клеро. Первая функция должна проверять, является ли заданное уравнение уравнением Клеро относительно заданной неизвестной функции, и возвращать в качестве результата функцию $\varphi(p)$, если проверка оказалась положительной, и значение `None` в противном случае. Вторая функция должна выдавать общее и особое решения уравнения Клеро, заданного функцией $\varphi(p)$.

8 Пусть семейство прямых линий задано соотношением

$$y = A(p)x + B(p). \tag{14}$$

Для построения огибающей этого семейства не нужно каждый раз составлять уравнение Клеро и искать его особое решение. Эти выкладки можно сделать один раз и получить формулы огибающей в параметрическом виде, выраженные через функции $A(p)$ и $B(p)$:

$$x = -\frac{B'(p)}{A'(p)},\quad y = -\frac{B'(p)A(p)}{A'(p)} + B(p). \tag{15}$$

Примените эти формулы для построения огибающих семейств прямых из упражнения 4.

[3] Если выражение в самом деле линейно, в противном случае генерируется исключение.

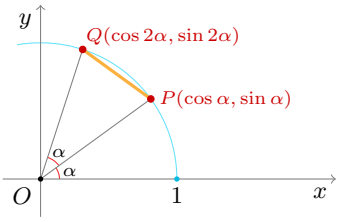


РИС. 8 Правило построения хорды PQ для случая $n = 2$

9 Рассмотрим задачу построения огибающей семейства хорд единичной окружности, соединяющих точки с полярными углами α и $n\alpha$, где n — фиксированный параметр (рис. 8). Координаты указанных точек равны

$$P(\cos\alpha, \sin\alpha) \text{ и } Q(\cos n\alpha, \sin n\alpha). \tag{16}$$

Записываем уравнение прямой, проходящей через эти точки:

$$\frac{x-\cos\alpha}{\cos 2\alpha - \cos\alpha} = \frac{y-\sin\alpha}{\sin 2\alpha - \sin\alpha}. \tag{17}$$

Разрешаем последнее равенство относительно y и приходим к уравнению семейства прямых (по параметру α)

$$y = \frac{\sin 2\alpha - \sin\alpha}{\cos 2\alpha - \cos\alpha}\cdot x - \frac{\sin\alpha}{\cos 2\alpha - \cos\alpha}. \tag{18}$$

Примените к построенному семейству формулы (15) для получения его огибающей. Постройте графики семейств хорд и соответствующих им огибающих. Если $n > 0$, то огибающей будет эпициклоида — кривая, образуемая фиксированной точкой окружности радиуса r, катящейся без скольжения по *внешней* стороне другой окружности радиуса $R = (n-1)r$. При $n = 2$ огибающая называется кардиоидой, при $n = 3$ — нефроидой. Несколько примеров огибающих-эпициклоид показано на рис. 9.

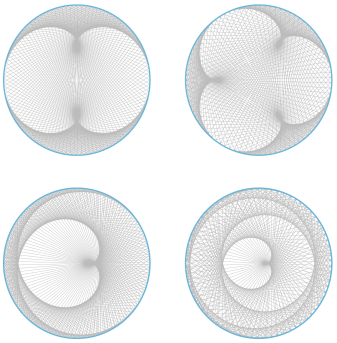


РИС. 9 Огибающие в форме эпициклоид

10 Если в предыдущем упражнении рассмотреть случай отрицательных значений параметра n, то получим в качестве огибающих так называемые гипоциклоиды. *Гипоциклоида* — это кривая, образуемая фиксированной точкой окружности, катящейся без скольжения по *внутренней* стороне другой окружности. Огибающие в форме гипоциклоид оказываются расположенными вне единичной окружности, на которой они строятся описанным методом, поэтому для их визуализации надо строить не хорды, а прямые, содержащие эти хорды (рис. 10).

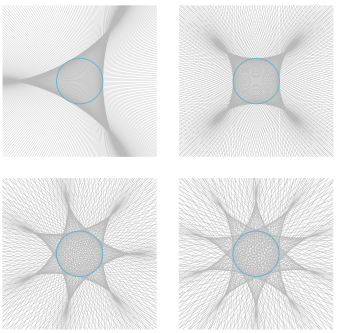


РИС. 10 Огибающие в форме гипоциклоид

11 Огибающую семейства *кривых* $F(x, y, C) = 0$, если она существует, можно найти исключением параметра C из алгебраической системы

$$F(x, y, C) = 0,\ F'_C(x, y, C) = 0. \tag{19}$$

Примените этот метод к построению огибающих следующих семейств кривых:

1) $y = x - \dfrac{(x-C)^2}{4}$;
2) $(x-C)^2 + (y-C)^2 = 1$;
3) $y^2 = 2Cx - C^2$;
4) $\dfrac{x^2}{C} + \dfrac{y^2}{1-C} = 1$ (см. рис. 11).

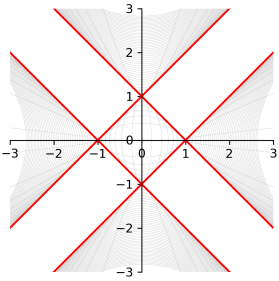


РИС. 11 Пример построения огибающих семейства кривых

12 Дифференциальное уравнение вида

$$y = x\varphi(y') + \psi(y') \tag{20}$$

называется уравнением Лагранжа. Решается это уравнение, как и уравнение Клеро, введением параметра $y' = p$, тогда

$$y = x\varphi(p) + \psi(p). \tag{21}$$

Дифференцируем это выражение по x, считая, что p — функция от x, и заменяем y' слева на p[4]:

$$p = \varphi(p) + x\varphi' p' + \psi' p' \Rightarrow p - \varphi(p) = (x\varphi' + \psi')p'. \tag{22}$$

Жозеф Луи Лагранж

Если p_0 — корень левой части $p - \varphi(p) = 0$, то $p = p_0$ является решением последнего уравнения. Это дает нам (возможно, особое) решение исходного уравнения

$$y = x\varphi(p_0) + \psi(p_0). \tag{23}$$

Если $p \neq \varphi(p)$, то переходим в (22) к обратной функции $x(p)$, заменяя dp/dx на $1/(dx/dp)$:

$$(p - \varphi(p))x' = x\varphi' + \psi'. \tag{24}$$

Получили линейное дифференциальное уравнение первого порядка относительно $x(p)$. Пусть его общее решение равно $x(p) = F(p, C)$. Подставляем это выражение в (21) и находим, что $y(p) = F(p, C)\varphi(p) + \psi(p)$. Вместе последняя пара функций определяет общее решение уравнения Лагранжа в параметрической форме:

$$x = F(p, C),\ y = F(p, C)\varphi(p) + \psi(p). \tag{25}$$

Решите с помощью SymPy по описанной схеме следующие дифференциальные уравнения и постройте графики всех их решений:

1) $y + xy' + y'^2 = 0$;
2) $2yy' + 4y = xy'^2$;
3) $y = xy'(y' + 2)$;
4) $y = xy'^2 + y'^2$.

[4] Производные φ' и ψ' вычисляются по p, а p' — по x.

ГЛАВА 10

Криволинейные зеркала

Задача • Рассмотрим следующую задачу: требуется найти кривую, после отражения от которой заданный пучок параллельных лучей сходится (фокусируется) в одной общей точке, называемой точкой фокуса. Пусть исходный пучок состоит из *вертикальных* лучей. Введем систему координат, ось Oy которой направим против направления лучей, тогда лучи будут падать на искомую кривую сверху вниз. Будем считать, что точка фокуса находится в начале координат, т. е после отражения все лучи должны пройти через точку O (рис. 1).

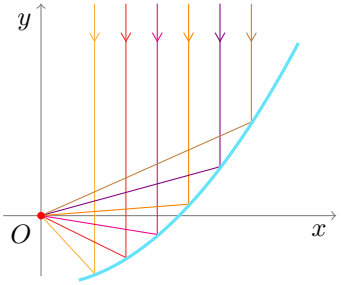


РИС. 1 Фокусирование отраженных от кривой лучей в начале координат

Пусть искомая кривая задана функцией $y(x)$. Рассмотрим траекторию выделенного луча MAO с абсциссой x (рис. 2). Его отражение происходит в точке $A(x, y)$. По закону отражения угол падения $\angle MAN$ равен углу отражения $\angle OAB$. Следовательно,

$$\angle OAB = \angle MAN = \angle BAC. \qquad (1)$$

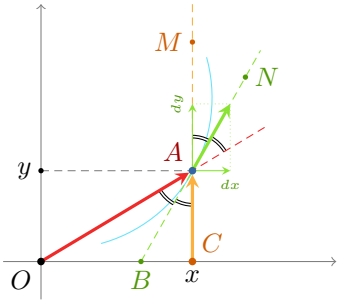


РИС. 2 Отражение луча MA от искомой кривой

Далее воспользуемся векторным способом. Условие (1) означает, что угол между векторами OA и BA равен углу между векторами BA и CA. Координаты вектора OA находятся элементарно: $OA = (x, y)$. Далее заметим, что прямая BA является касательной к $y(x)$, поэтому вектор BA должен быть сонаправлен вектору $\delta = (dx, dy)$ (см. рис. 2). Следовательно, при сравнении углов между векторами мы можем заменить вектор BA на вектор δ. А вместо вектора CA возьмем сонаправленный с ним единичный вектор $e = (0, 1)$.

Теперь применим формулу для косинуса угла меж-

ду двумя векторами:

$$\frac{OA \cdot \delta}{|OA| \cdot |\delta|} = \frac{e \cdot \delta}{|e| \cdot |\delta|}. \tag{2}$$

Длина вектора OA равна $\sqrt{x^2 + y^2}$, вектор e единичный. Скалярные произведения вычислим, используя известные координаты векторов:

$$OA \cdot \delta = xdx + ydy,\ e \cdot \delta = dy.$$

Подставляя все эти соотношения в (2), получаем дифференциальное уравнение, решение которого должно описывать искомую кривую:

$$\frac{xdx + ydy}{\sqrt{x^2 + y^2}} = dy. \tag{3}$$

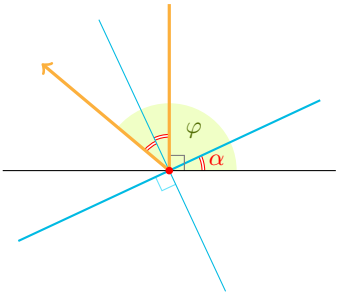


РИС. 3 Вычисление угла наклона отраженного луча

Модель ● Решим поставленную задачу в `SymPy`. Проверим найденное решение с помощью кода, который строит семейство отраженных лучей от заданной кривой. Дополнительно рассмотрим понятие каустики отражения как огибающей кривой семейства прямых линий, представляющих лучи света.

1 Подключаем стандартный набор библиотек.

2 Сначала выведем уравнение семейства лучей, отраженных от некоторой кривой. Для этого рассмотрим плоскость OXY, на этой плоскости — кривую $Y = y(X)$. Выберем на кривой точку с координатами $(x, y(x))$ и рассмотрим луч, падающий в эту точку вертикально сверху вниз. Отражение луча происходит относительно касательной к данной кривой в рассматриваемой точке. Пусть угол наклона касательной к оси OX равен α, т. е.

$$k = \operatorname{tg}\alpha = y'(x). \tag{4}$$

Несложно показать (см. рис. 3), что отраженный от такой касательной луч будет наклонен к оси OX под углом $\varphi = 90^\circ + 2\alpha$, тогда

$$\operatorname{tg}\varphi = \operatorname{tg}(90^\circ + 2\alpha) = -\operatorname{ctg} 2\alpha = \\ = \frac{\operatorname{tg}^2\alpha - 1}{2\operatorname{tg}\alpha} = \frac{k^2 - 1}{2k}. \tag{5}$$

Таким образом, прямая, содержащая отраженный луч, наклонена к OX под углом φ и проходит через точку с координатами $(x, y(x))$, следовательно, ее уравнение имеет вид

$$Y = \operatorname{tg}\varphi \cdot (X - x) + y(x) = Ax + B, \tag{6}$$

где

$$A = \operatorname{tg}\varphi = \frac{k^2 - 1}{2k},\ B = y(x) - Ax. \tag{7}$$

Так как k зависит от x, то уравнение (6) представляет собой искомое параметрическое описание семейства отраженных лучей, в котором роль параметра играет символ x. Повторим теперь эти выкладки в SymPy. Определяем символы X и Y для описания семейства прямых, параметр x и функцию $y(x)$.

```
X, Y, x = symbols("X Y x")
y = Function("y")(x)
```

3 Вычисляем производную $k = y'(x)$ и задаем коэффициенты A и B согласно формулам (7).

```
k = diff(y, x)
A = (k ** 2 - 1) / (2 * k)
B = simplify(y - x * A)
display(A, B)
```

$$\frac{\left(\frac{d}{dx}y(x)\right)^2 - 1}{2\frac{d}{dx}y(x)}$$
$$-\frac{x\frac{d}{dx}y(x)}{2} + \frac{x}{2\frac{d}{dx}y(x)} + y(x)$$

4 Записываем уравнение семейства отраженных лучей (6).

```
fam = Eq(Y, A * X + B)
display(fam)
```

$$Y = \frac{X\left(\left(\frac{d}{dx}y(x)\right)^2 - 1\right)}{2\frac{d}{dx}y(x)} - \frac{x\frac{d}{dx}y(x)}{2} + \frac{x}{2\frac{d}{dx}y(x)} + y(x)$$

5 Для демонстрации полученных формул построим семейство лучей, отраженных от нижней единичной полуокружности. Задаем уравнение `f1` такой кривой и подставляем его вместо `f` в уравнение `fam`. Метод `doit` применяется в данном случае для того, чтобы SymPy вычислил все производные.

```
f1 = -sqrt(1 - x ** 2)
fam1 = simplify(fam.subs(y, f1).doit())
display(fam1)
```

$$Y = \frac{2Xx^2 - X - x}{2x\sqrt{1 - x^2}}$$

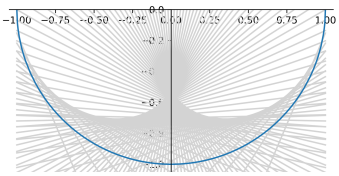


РИС. 4 Семейство лучей, отраженных от полуокружности

6 Строим график полученного семейства `fam1` и поверх него исходную полуокружность (рис. 4).

```
p1 = plot(xlim = (-1, 1), ylim = (-1, 0),
          aspect_ratio = (1, 1), show = False)
for x0 in np.arange(-0.98, 1, 0.02):
    l = fam1.rhs.subs(x, x0)
    p = plot(l, (X, -1, 1), line_color = "gray",
             show = False)
    p1.extend(p)
p1.extend(plot(f1, (x, -1, 1), show = False))
p1.show()
```

Внутри окружности отчетливо различима кривая, образованная сгущением прямых линий. Подобные кривые в оптике называются *каустиками*. Каустика — это особая линия или поверхность, вблизи которой резко возрастает интенсивность освещения за счет отражения или преломления потока световых лучей некоторой поверхностью или телом (рис. 5). Математически каустика представляет собой огибающую семейства прямых, поэтому мы можем использовать результаты предыдущей главы для нахождения уравнения каустики в нашем случае.

РИС. 5 Каустика отражения

7 Так как у нас уже имеется уравнение семейства прямых, то для построения их огибающей воспользуемся формулами (15) из упражнения 8 предыдущей главы. Заменим в выражениях `A` и `B` символ функции `f` на уравнение полуокружности `f1` и вычислим согласно указанным формулам параметрическое уравнение искомой огибающей[1].

```
A1 = A.subs(y, f1).doit()
B1 = B.subs(y, f1).doit()
XC = simplify(- diff(B1, x) / diff(A1, x))
YC = simplify(A1 * XC + B1)
display(XC, YC)
```

$$x^3$$

$$-\sqrt{1 - x^2}\left(x^2 + \frac{1}{2}\right)$$

[1] Заметим, что в качестве параметра сейчас выступает символ `x`.

8 Добавим найденную каустику `(XC, YC)` к построенному ранее графику `p1` (рис. 6).

```
p2 = plot(xlim = (-1, 1), ylim = (-1, 0),
          aspect_ratio = (1, 1), show = False)
p = plot_parametric((XC, YC), (x, -1, 1),
                    line_color = "red",
                    show = False)
p2.extend(p1)
p2.extend(p)
p2.show()
```

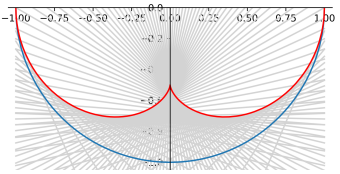


РИС. 6 Каустика отражения для полуокружности

Отметим, что найденная нами параметрическая кривая называется нефроидой и представляет собой частный случай эпициклоиды[2].

9 Вернемся теперь к основной задаче — нахождению кривой $y(x)$, при отражении от которой все лучи фокусируются в одной общей точке. Будем следовать описанной выше схеме. Определяем символы для дифференциалов[3] dx и dy. Строим три вектора OA, δ и e, используя кортежи `SymPy`.

```
dx, dy = symbols("dx dy")
OA = Tuple(x, y)
delta = Tuple(dx, dy)
e = Tuple(0, 1)
display(OA, delta, e)
```

$$(x,\ y(x))\ \ (dx,\ dy)\ \ (0,\ 1)$$

10 Определяем функцию для скалярного произведения двух двумерных векторов. Тестируем ее на вычислении произведения $OA \cdot \delta$.

```
def dot(u, v):
    return u[0] * v[0] + u[1] * v[1]
display(dot(OA, delta))
```

$$dx\,x + dy\,y(x)$$

11 Определяем функцию вычисления длины заданного вектора, тестируем ее на векторе OA.

```
def length(u):
    return sqrt(dot(u, u))
display(length(OA))
```

$$\sqrt{x^2 + y^2(x)}$$

12 Составляем искомое дифференциальное уравнение, используя равенство (2).

[2] См. упражнение 9 предыдущей главы.

[3] Библиотека `SymPy` не умеет работать с дифференциалами, поэтому нам придется вводить их вручную просто как символы.

```
ode = Eq(dot(OA, delta) / length(OA),
         dot(e, delta) / length(e))
display(ode)
```

$$\frac{dx\,x + dy\,y(x)}{\sqrt{x^2 + y^2(x)}} = dy$$

13 Для перехода от уравнения в дифференциалах к уравнению с производной формально нужно выполнить деление всего уравнения на dx и заменить отношение дифференциалов dy/dx на производную y'. Однако того же результата можно добиться простой заменой dx на 1, а dy — на y'.

```
ode1 = ode.subs({dx: 1, dy: diff(y, x)})
display(ode1)
```

$$\frac{x + y(x)\frac{d}{dx}y(x)}{\sqrt{x^2 + y^2(x)}} = \frac{d}{dx}y(x)$$

14 Проверим тип уравнения. Вторая строка вывода показывает, что это уравнение в полных дифференциалах[4].

[4] См. упражнения 1 и 2.

```
classify_ode(ode1, y)
```

```
factorable
1st_exact
1st_homogeneous_coeff_best
1st_homogeneous_coeff_subs_indep_div_dep
1st_homogeneous_coeff_subs_dep_div_indep
1st_power_series
lie_group
1st_exact_Integral
1st_homogeneous_coeff_subs_indep_div_dep_Integral
1st_homogeneous_coeff_subs_dep_div_indep_Integral
```

15 Решаем уравнение `ode1` как уравнение в полных дифференциалах, используя подсказку `1st_exact`.

```
dsol = dsolve(ode1, y, hint = "1st_exact")
display(dsol)
```

$$y(x) = -\frac{C_1}{2} + \frac{x^2}{2C_1}$$

16 Как видно, найденное решение оказалось квадратичной функцией, следовательно, искомыми кривыми, фокусирующими все отраженные лучи в одной точке, являются параболы. Построим семейство таких парабол (см. рис. 7). Так как все они имеют общую точку фокуса, то данное семейство называется семейством конфокальных парабол[5].

```
C1 = symbols("C1")
p3 = plot(xlim = (-5, 5), ylim = (-5, 5),
          aspect_ratio = (1, 1),
          show = False)
for c in np.arange(-10, 10, 0.5):
    p = plot(dsol.rhs.subs(C1, c), (x, -5, 5),
             show = False)
    p3.extend(p)
p3.show()
```

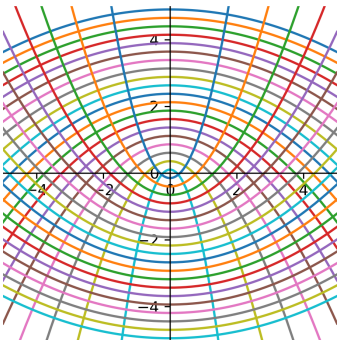


РИС. 7 Семейство конфокальных парабол

17 Выберем одну из найденных парабол (например, для случая $C_1 = 1$) и построим семейство отраженных от нее лучей, чтобы проверить, что все эти лучи в самом деле проходят через начало координат. Определим сначала уравнение данной параболы и сохраним его в переменной `f2`.

```
f2 = dsol.rhs.subs(C1, 1)
display(f2)
```

$$\frac{x^2}{2} - \frac{1}{2}$$

18 Подставим эту формулу в выведенное выше общее уравнение `fam` семейства отраженных лучей.

```
fam2 = fam.subs(y, f2).doit()
display(fam2)
```

$$Y = \frac{X\left(x^2-1\right)}{2x}$$

19 Построим на одном графике параболу `f2` и порожденное ею семейство прямых `fam2`.

```
p4 = plot(xlim = (-1, 1), ylim = (-1, 1),
          aspect_ratio = (1, 1), show = False)
for x0 in np.arange(-0.95, 1, 0.05):
    l = fam2.rhs.subs(x, x0)
    p = plot(l, (X, -1, 1), line_color = "gray",
             show = False)
    p4.extend(p)
p4.extend(plot(f2, (x, -1, 1), show = False))
p4.show()
```

[5] См. упражнение 5.

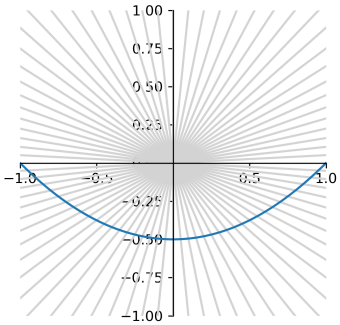


РИС. 8 Семейство отраженных от параболы лучей

Результат построения, наглядно показывающий фокусирование отраженных лучей в начале координат, приведен на рис. 8.

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Определите в `SymPy` следующие уравнения, преобразуйте их к форме с производной, проверьте с помощью команды `classify_ode`, что данные уравнения являются уравнениями в полных дифференциалах, и решите их, используя подсказку `1st_exact`, без упрощения результата (опция `simplify = False`). Сравните найденное решение с решениями, полученными с помощью других подсказок.

1) $2xydx + (x^2 - y^2)dy = 0$;
2) $(2 - 9xy^2)xdx + (4y^2 - 6x^3)ydy = 0$;
3) $e^{-y}dx - (2y + xe^{-y})dy = 0$;
4) $(1 + y^2 \sin 2x)dx - 2y\cos^2 xdy = 0$.

2 Известно, что уравнение

$$P(x, y)dx + Q(x, y)dy = 0 \tag{8}$$

является уравнением в полных дифференциалах, если для него выполнено условие

$$\frac{\partial P}{\partial y} = \frac{\partial Q}{\partial x}. \tag{9}$$

Общее решение уравнения (8) можно найти, используя следующую формулу[6]:

$$\int_{x_0}^{x} P(x, y)dx + \int_{y_0}^{y} Q(x_0, y)dy = C, \tag{10}$$

где (x_0, y_0) — произвольная точка в области определения функций P и Q. Напишите две функции, первая из которых проверяет, является ли заданное уравнение (в дифференциалах) уравнением в полных дифференциалах или нет, а вторая — решает это уравнение описанным выше способом. Протестируйте эти функции на уравнениях из предыдущего упражнения.

[6] Обратите внимание, что во втором интеграле вместо x используется x_0!

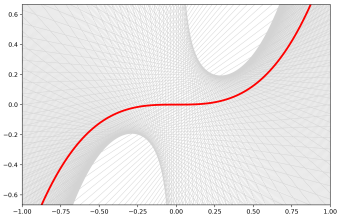


РИС. 9 Каустика, порожденная кривой $y = x^3$ (упражнение 2)

3 Постройте каустику отражения, порожденную заданной кривой $y(x)$, для случая параллельного пучка лучей, падающих на эту кривую сверху вниз.

1) $y(x) = x^3$ (см. рис. 9);
2) $y(x) = x^4$;
3) $y(x) = \sin x$;
4) $y(x) = \sqrt{1 + x^2}$.

4 Постройте каустику отражения вертикальных лучей, порожденную полученным при построении модели уравнением параболы `f2`.

5 Рассмотрим семейство конфокальных парабол

$$y = -\frac{C}{2} + \frac{x^2}{2C}. \tag{11}$$

Покажите, что два его подсемейства, различающихся знаком параметра C, являются взаимно ортогональными друг другу (см. рис. 7).

6 Дифференциальное уравнение, описывающее полученное нами семейство парабол `dsol`, можно вывести и другим способом. У нас есть уравнение семейства отраженных лучей $Y = AX + B$, порожденных кривой $y(x)$. Потребуем, чтобы все эти прямые проходили через начало координат, для этого необходимо и достаточно, чтобы коэффициент B равнялся нулю. Так как формулы для обоих коэффициентов A и B содержат в себе производную y', то условие $B = 0$ оказывается искомым дифференциальным уравнением:

```
display(Eq(B, 0))
```

$$-\frac{x\frac{d}{dx}y(x)}{2} + \frac{x}{2\frac{d}{dx}y(x)} + y(x) = 0$$

В такой форме дифференциальное уравнение не является уравнением в полных дифференциалах. Однако если разрешить его относительно $y(x)$, то увидим, что данное уравнение является уравнением Лагранжа. Решите его методом, описанным в упражнении 12 предыдущей главы.

7 Обобщите решенную в главе задачу на случай фокусирования отраженных лучей в произвольной точке плоскости.

8 Обобщите и решите рассмотренную в главе задачу на случай, когда исходный пучок параллельных лучей наклонен к оси OX под заданным углом γ.

9 Решите по описанной в упражнении 6 схеме следующую задачу: определить форму кривой, после отражения от которой вертикально падающие на нее лучи оказываются параллельными заданной прямой $Y = aX$.

10 Выполните вывод уравнения каустики в случае, если лучи, падающие на внутреннюю часть единичной окружности с центром в начале координат, исходят из нижней точки этой окружности (рис. 10). Каустикой в данном случае оказывается еще один эпициклоида, называемая *кардиоидой*.

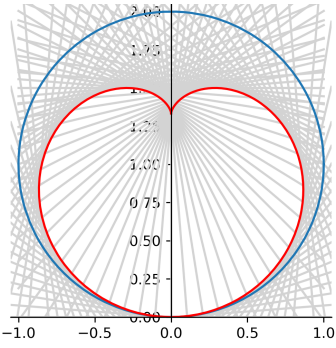


РИС. 10 Каустика отраженных от окружности лучей, исходящих из нижней точки этой окружности

11 Выведите уравнение каустики для общего случая, когда лучи, исходящие из начала координат, падают снизу на заданную кривую $y(x)$. Примените полученную формулу для получения каустик, порожденных следующими кривыми:

1) $y(x) = 1 - x^2$;
2) $y(x) = (x-1)^2(x+1)$;
3) $y(x) = \cos x$;
4) $y(x) = \ln(1+x^2)$.

12 Решите задачу, обратную к рассмотренной в данной главе: найти форму кривой, после отражения от которой все лучи, исходящие из начала координат, оказываются направленными вертикально вверх.

13 Поставьте и решите с помощью SymPy задачу нахождения такой кривой $y(x)$, после отражения от которой все лучи, исходящие из заданной точки F_1, проходят через другую заданную точку F_2 (см. рис. 11).

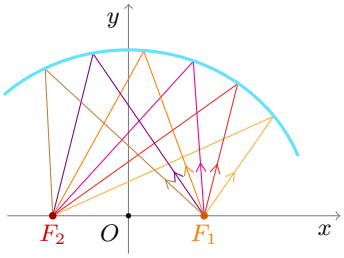


РИС. 11 Фокусирование после отражения от кривой лучей, исходящих из точки F_1, в точке F_2

14 Поставьте и решите с помощью SymPy задачу нахождения кривой $y(x)$, после отражения от которой все лучи, направленные в заданную точку F_2, фокусируются во второй точке F_1 (см. рис. 12).

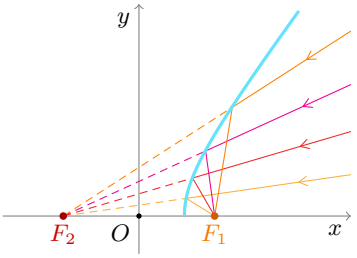


РИС. 12 Фокусирование после отражения от кривой лучей, направленных в точку F_2, в точке F_1

ГЛАВА 11

Из пушки на Луну

Задача • В данной главе мы рассмотрим следующую классическую гравитационную задачу: некоторый снаряд выстреливается с заданной начальной скоростью v_0 с поверхности Земли вертикально вверх в направлении Луны, требуется определить минимальную начальную скорость v_0, которую необходимо придать снаряду, чтобы он достиг поверхности Луны.

Предполагается, что Земля и Луна неподвижны и находятся на фиксированном расстоянии a друг от друга[1]. При решении этой задачи нами не будут учитываться никакие другие факторы и силы, кроме сил притяжения, действующих на снаряд со стороны Земли и Луны. В частности, мы пренебрежем силой сопротивления воздуха, кстати, весьма значительной с точки зрения практического решения данной задачи. Физические характеристики рассматриваемой астрономической системы (G — гравитационная постоянная, R и M — радиус и масса Земли, r и m — радиус и масса Луны, a — расстояние между их центрами), необходимые для решения поставленной задачи, приведены в табл. 1.

[1] То есть a — это расстояние между центрами этих двух тел.

ТАБЛ. 1 Физические характеристики задачи

G	$6.67 \cdot 10^{-11}$ м^3/с$^2\cdot$кг
R	$6.38 \cdot 10^{6}$ м
M	$6 \cdot 10^{24}$ кг
r	$1.73 \cdot 10^{6}$ м
m	$7.3 \cdot 10^{22}$ кг
a	$4 \cdot 10^{8}$ м

Введем систему координат, ее начало O совместим с центром Земли, а единственную ось Ox (движение в нашей задаче очевидно является одномерным) направим на Луну. То есть снаряд начинает движение из точки $x = R$, а закончить движение должен в точке $x = a - r$ (рис. 1). Пусть масса снаряда равна m_0. Тогда на него действуют две силы тяжести F_1 и F_2 со стороны Земли и Луны соответственно:

$$F_1 = -\frac{GMm_0}{x^2},\ F_2 = \frac{Gmm_0}{(a-x)^2}. \qquad (1)$$

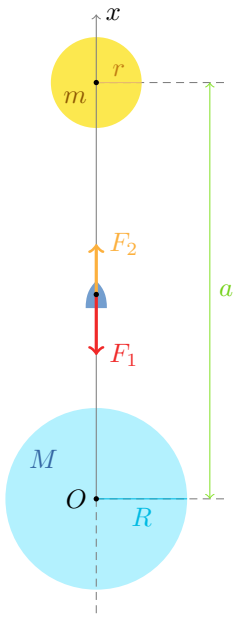


РИС. 1 Конфигурация модели

Записываем второй закон Ньютона, в котором $\ddot{x}$ означает ускорение снаряда. После сокращения на m_0 получаем дифференциальное уравнение второго порядка для неизвестной функции $x(t)$:

$$\ddot{x} = -G\left(\frac{M}{x^2} - \frac{m}{(a-x)^2}\right). \tag{2}$$

Начальные условия для уравнения (2):

$$x(0) = R,\ \dot{x}(0) = v_0. \tag{3}$$

Так как уравнение (2) не содержит в явном виде независимой переменной t, то его порядок можно понизить заменой неизвестной функции $x(t)$ на функцию $v(x) = \dot{x}$. Тогда

$$\ddot{x} = \frac{d(\dot{x})}{dt} = \frac{dv}{dt} = \frac{dv}{dx}\frac{dx}{dt} = v'v, \tag{4}$$

и мы приходим к уравнению первого порядка

$$v \cdot v' = -G\left(\frac{M}{x^2} - \frac{m}{(a-x)^2}\right). \tag{5}$$

Согласно условиям (3) в начальный момент времени $x(0) = R$, $v(0) = v_0$, следовательно, начальным условием для уравнения (5) будет условие

$$v(R) = v_0. \tag{6}$$

Модель • Решим поставленную задачу с помощью SymPy. Помимо ответа на вопрос о минимальной начальной скорости v_0, необходимой для достижения поверхности Луны, мы найдем и общую зависимость скорости снаряда от его расстояния до центра Земли для заданной скорости v_0.

1 Подключаем библиотеки SymPy и NumPy.

2 Определяем символ t и функцию $x(t)$.

3 Вводим символы для всех параметров задачи, перечисленных в табл. 1.

4 Определяем словарь со значениями введенных в предыдущем пункте параметров.

```
par = {G: 6.67 * 10 ** -11, M: 5.97 * 10 ** 24,
       m: 7.36 * 10 ** 22, R: 6.380 * 10 ** 6,
       r: 1.730 * 10 ** 6, a: 4.0 * 10 ** 8}
print(par)
```

```
G: 6.67e-11, M: 5.969999999999999e+24, m: 7.36e+22,
R: 6380000.0, r: 1730000.0, a: 400000000.0
```

5 Определяем силы F_1 и F_2 согласно формулам (1), предполагая[2], что $m_0 = 1$.

[2] Дифференциальное уравнение и его решение от массы снаряда в данной задачи никак не зависят.

```
m0 = 1
F1 = - G * M * m0 / x ** 2
F2 = G * m * m0 / (a - x) ** 2
display(F1, F2)
```

$$-\frac{GM}{x^2(t)}$$
$$\frac{Gm}{(a - x(t))^2}$$

6 Записываем исходное дифференциальное уравнение второго порядка (2).

```
ode = Eq(m0 * diff(x, t, t), F1 + F2)
display(ode)
```

$$\frac{d^2}{dt^2}x(t) = -\frac{GM}{x^2(t)} + \frac{Gm}{(a - x(t))^2}$$

7 Несложно убедиться, что полученное нами нелинейное дифференциальное уравнение второго порядка библиотекой SymPy не решается[3]. Применим к нему описанную выше замену $x(t) \rightsquigarrow v(x)$ для понижения порядка уравнения. Так как функция $x(t)$ после замены становится переменной x, то по правилам SymPy мы должны ввести для этой переменной новый символ. Определяем функцию v от этой переменной и создаем словарь с тремя ключами x, $\dot{x}$ и $\ddot{x}$, соответствующий выполняемой замене.

[3] Это уравнение вообще не решается аналитически.

```
X = symbols("x")
v = Function("v")(X)
Z = {diff(x, t, t): diff(v, X) * v,
     diff(x, t): v,
     x: X}
print(Z)
```

```
Derivative(x(t), t, 2): v(x) * Derivative(v(x), x),
Derivative(x(t), t): v(x), x(t): x
```

8 Выполняем замену, применяя подстановку `Z` к уравнению `ode`.

```
ode2 = ode.subs(Z)
display(ode2)
```

$$v(x)\frac{d}{dx}v(x) = -\frac{GM}{x^2} + \frac{Gm}{(a-x)^2}$$

9 Вводим символ начальной скорости v_0, определяем с его помощью начальное условие (6) и решаем поставленную начальную задачу в неявной форме.

```
v0 = symbols("v0", positive = True)
ic = {v.subs(X, R): v0}
dsol = dsolve(ode2, v, ics = ic, simplify = False)
display(dsol)
```

$$\frac{v^2(x)}{2} = -\frac{GMa + x\,(-GM + Gm)}{-ax + x^2} + \\ + \frac{2GMR - 2GMa - 2GRm - R^2v_0^2 + Rav_0^2}{-2R^2 + 2Ra}$$

10 Для нахождения минимальной начальной скорости, при которой снаряд долетит до поверхности Луны, надо проанализировать скорость снаряда в так называемой *точке либрации* — точке равенства модулей сил F_1 и F_2. Несложно заметить, что если снаряд в точке либрации имеет положительную скорость, то он обязательно достигнет Луны. Критическим, очевидно, будет случай, когда в точке либрации скорость снаряда будет в точности равна нулю[4]. Сначала составим и решим уравнение для нахождения расстояния L от центра Земли до точки либрации.

[4] Теоретически это будет означать, что снаряд останется в этой точке навсегда. На практике, однако, любое малое возмущение положения снаряда относительно точки либрации приведет к падению снаряда или на Землю, или на Луну.

```
eq = Eq(F1 + F2, 0)
sol = solve(eq, x)
display(sol)
```

```
[a*(M-sqrt(M*m))/(M-m), a*(M+sqrt(M*m))/(M-m)]
```

11 Как видно, мы получили два возможных решения для расстояния L, однако нам подходит только одно из них — то, которое меньше a[5]. Подставим в оба найденных решения набор параметров `par` и сравним полученные значения с расстоянием a (из того же набора параметров).

[5] Объясните, почему вторая точка не является точкой либрации.

```
for s in sol:
    print(s.subs(par))
print(par[a])
```

```
360025337.331458
449960416.687944
400000000.0
```

12 То есть искомому значению L соответствует выражение `sol[0]`. Запомним его в переменной `L`. В переменной `l` сохраним значение этого выражения, вычисленное исходя из условий нашей задачи.

```
L = sol[0]
l = float(L.subs(par))
display(L, l)
```

$$\frac{a\left(M-\sqrt{Mm}\right)}{M-m}$$

```
360025337.331458
```

13 Теперь мы можем найти искомую начальную скорость. Для этого заменим в решении `dsol` скорость `v` на ноль, а расстояние `X` — на `L`. Полученное таким образом уравнение решим относительно символа `v0` и в качестве ответа выберем[6] положительную величину.

```
VE = solve(dsol.subs({v: 0, X: L}), v0)[1]
display(VE)
```

$$\sqrt{2}\sqrt{-\frac{G\left(M^2R^2\sqrt{Mm}+\cdots+a^2\left(Mm\right)^{\frac{3}{2}}\right)}{Ra\left(R-a\right)\left(-2Mm+M\sqrt{Mm}+m\sqrt{Mm}\right)}}$$

14 Вычислим значение критической начальной скорости (в метрах в секунду) для нашей задачи.

```
ve = float(VE.subs(par))
print(ve)
```

```
11063.20675112643
```

Интересно, что найденная величина скорости всего на 110 м/с (т. е. на 1 %) меньше второй космической скорости для Земли[7].

[6] Полученное выражение слишком велико, поэтому в выводе показана только небольшая его часть.

[7] См. упражнение 8.

15 Выразим из найденного решения `dsol` скорость `v`, взяв положительный корень уравнения, что соответствует движению снаряда вверх, и подставим в полученное выражение параметры `par`. Построим графики найденной зависимости скорости снаряда от его положения для разных значений начальной скорости v_0 (рис. 2). По оси абсцисс графики будем строить от точки на поверхности Земли ($x = R$) до точки на поверхности Луны ($x = a - r$). Для скоростей, меньших критической, цвет кривых сделаем синим, для критической скорости — красным, а для скоростей, больших критической, — зеленым.

```
ds = solve(dsol, v)[1].subs(par)
p1 = plot(ylim = (0, 10000),
          size = (6, 8), show = False)
xlim = (X, par[R], par[a] - par[r])
for u in np.arange(ve - 2100, ve + 2100, 70):
    if u < ve: c = "blue"
    elif u > ve: c = "green"
    else: c = "red"
    p = plot(ds.subs(v0, u), xlim,
             line_color = c, show = False)
    p1.extend(p)
p1.show()
```

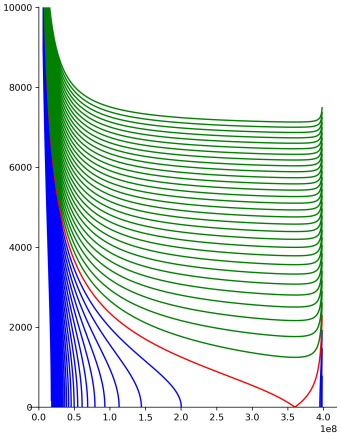


РИС. 2 Зависимость $v(x)$ скорости снаряда от его положения для разных значений начальной скорости v_0

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Оцените минимальную скорость, с которой снаряд, выпущенный с поверхности Земли, упадет на поверхность Луны.

2 Чтобы довести до конца решение исходного уравнения `ode`, надо применить к найденному решению `dsol` обратную замену. Это можно сделать, используя словарь `Z`, поменяв в нем местами ключи и значения элементов:

```
Z2 = {v: k for k, v in Z.items()}
ode2 = dsol.subs(Z2)
display(ode2)
```

$$\frac{\left(\frac{d}{dt}x(t)\right)^2}{2} = \frac{-GMa - (-GM + Gm)\,x(t)}{-ax(t) + x^2(t)} + \\ + \frac{2GMR - 2GMa - 2GRm - R^2v_0^2 + Rav_0^2}{-2R^2 + 2Ra}$$

Выразите из найденного соотношения производную $\dot{x}$ и решите приближенно полученное дифференциальное уравнение относительно $x(t)$ с начальным условием $x(0) = R$.

С помощью этого решения найдите время полета снаряда до поверхности Луны для заданной начальной скорости v_0.

3 Если начальная скорость снаряда невелика (например, $v_0 = 10$), то высота его полета будет небольшой и поэтому силой притяжения со стороны Луны можно пренебречь. При таких условиях снаряд должен двигаться практически равноускоренно с ускорением g:

$$x(t) = v_0 t - \frac{gt^2}{2}. \tag{7}$$

Сравните численное решение уравнения `ode`, построенное по схеме решения предыдущего упражнения, с приближенной оценкой (7) для $v_0 = 10$.

4 Численно можно решать и исходное уравнение (2), преобразовав его предварительно в систему двух дифференциальных уравнений первого порядка относительно двух неизвестных функций $x(t)$ и $v(t)$:

$$\begin{aligned} \dot{x} &= v, \\ \dot{v} &= -G\left(\frac{M}{x^2} - \frac{m}{(a-x)^2}\right). \end{aligned} \tag{8}$$

Преимуществом такого подхода является то, что не нужно отдельно рассматривать случаи, когда тело движется вверх ($\dot{x} > 0$) и вниз ($\dot{x} < 0$), как это делалось в упражнении 2.

5 Постройте график зависимости максимальной высоты полета снаряда от его начальной скорости v_0.

6 После того как снаряд достигает наивысшей точки полета (при условии ее существования), его скорость становится отрицательной, т. е. он начинает падать обратно на Землю. Постройте и проанализируйте график зависимости скорости снаряда от его положения в области отрицательных скоростей (см. рис. 3).

7 Рассмотренная нами модель достижения Луны была положена в основу сюжета научно-фантастического романа Жюля Верна «С Земли на Луну прямым путём за 97 часов 20 минут»[8], написанного в 1865 году. Пушка, из которой выстреливался снаряд, имела длину ствола 274 метра. Если предположить, что снаряд в стволе пушки разгонялся равноускоренно, то простые расчеты показывают, что ускорение, необходимое для достижения найденной выше скорости v_0, должно быть примерно равным $23\,000g$. Заметим, что максимальная зафиксированная перегрузка при аварии гоночного автомобиля с *выжившим* водителем составляет $214g$, т. е. у пассажиров снаряда в романе Жюля Верна не было никаких шансов выжить при выстреле из

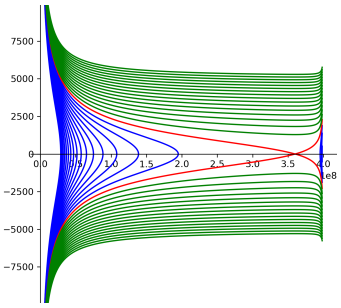


РИС. 3 Зависимость $v(x)$ с учетом отрицательных скоростей

Жюль Верн

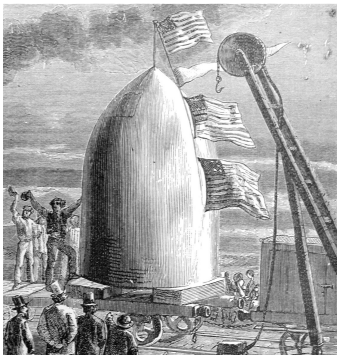


РИС. 4 Иллюстрация из первого издания романа «Из пушки на Луну»

[8] В русскоязычных переводах, изданных в СССР, известен также под названиями «Из пушки на Луну» и «От Земли до Луны».

пушки. Рассчитайте, какой должна была быть длина ствола пушки из романа Жюля Верна, чтобы нужная скорость достигалась с постоянным ускорением $10g$. Сравните эту длину с глубиной Кольской сверхглубокой скважины[9].

[9] В романе в качестве пушки использовалась шахта в земле с чугунными стенками диаметром 18.3 метра.

8 Постройте аналогичную модель движения снаряда в поле тяготения только Земли. Используйте найденное решение для вычисления второй космической скорости — минимальной начальной скорости, при которой снаряд никогда не упадет обратно на Землю. Рассчитайте с помощью полученной формулы величину второй космической скорости для различных тел Солнечной системы.

9 При уменьшении радиуса R тела, с которого выстреливается снаряд, при условии что масса тела остается постоянной и равной M, величина второй космической скорости неограниченно увеличивается и, начиная с некоторого критического значения R_0, становится больше скорости света c, т. е. тело превращается в *черную дыру*. Критическая величина радиуса R_0 называется *гравитационным радиусом*, или *радиусом Шварцшильда*, данного тела. Вычислите, используя результат предыдущего упражнения, гравитационный радиус Луны, Земли и Солнца.

Карл Шварцшильд

10 Решите с помощью рассмотренной выше (в главе и в упражнении 2) замены $z(y) = y'(x)$ следующие дифференциальные уравнения второго порядка :

1) $yy'' = y'^2 - y'^3$;
2) $y'^2 + 2yy'' = 0$;
3) $y'' + y'^2 = 0$;
4) $y'' = 2yy'$.

11 Порядок дифференциального уравнения вида

$$F(x, y', y'') = 0 \tag{9}$$

может быть понижен с помощью замены $z(x) = y'(x)$, применение которой приводит к уравнению первого порядка вида

$$F(x, z, z') = 0. \tag{10}$$

Определите в SymPy такую замену и решите с ее помощью следующие уравнения:

1) $2xy'y'' = y'^2 - 1$;
2) $x^2y'' = y'^2$;
3) $y'' = y' \operatorname{ctg} x$;
4) $x \ln x\, y'' = y' - 1$.

ГЛАВА 12
Метроном

Задача • Метроном — прибор, отмечающий короткие промежутки времени равномерными ударами. Механический метроном (рис. 1) представляет собой маятник с дополнительным грузом, закрепленным выше точки подвеса. Меняя положение этого груза, можно настраивать нужную частоту ударов метронома.

Рассмотрим (рис. 2) простую математическую модель метронома. Пусть основной груз имеет массу M и закреплен на невесомом стержне на расстоянии L ниже точки подвеса O, дополнительный груз имеет массу m и закреплен на том же стержне на высоте l выше точки подвеса. Очевидно, что каждый груз метронома совершает движение по дуге окружности своего радиуса с центром в точке крепления стержня. Поэтому положение обоих грузов в любой момент времени однозначно определяется углом θ отклонения стержня от вертикали.

Чтобы маятник не переворачивался, центр его масс должен располагаться относительно точки подвеса с той же стороны, что и основной груз:

$$ML > ml. \tag{1}$$

Вывод дифференциального уравнения для функции $\theta(t)$ проведем с помощью закона сохранения механической энергии. Пусть в момент времени t нижний груз находится в точке P, верхний — в точке Q. Потенциальная энергия системы равна сумме потенциальных энергий двух грузов, отсчитывать эту энергию будем относительно точки подвеса O:

$$E_{\text{П}} = mgl\cos\theta - MgL\cos\theta = (ml - ML)g\cos\theta. \tag{2}$$

РИС. 1 Механический метроном

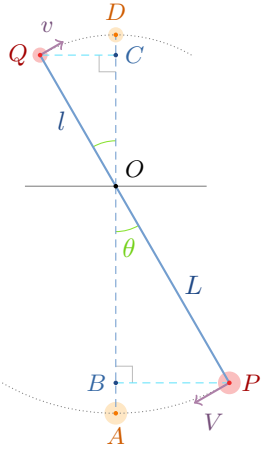


РИС. 2 Математическая модель метронома

Линейные скорости нижнего и верхнего грузов равны[1] $L\dot{\theta}$ и $l\dot{\theta}$ соответственно. Следовательно, кинетическая энергия всей системы вычисляется по формуле

$$E_{\text{К}} = \frac{ml^2\dot{\theta}^2}{2} + \frac{ML^2\dot{\theta}^2}{2} = \frac{(ML^2 + ml^2)}{2} \cdot \dot{\theta}^2. \quad (3)$$

В силу закона сохранения энергии:

$$E_{\text{К}} + E_{\text{П}} = E_0 = \text{const}, \quad (4)$$

где E_0 — полная механическая энергия рассматриваемой системы. Следовательно,

$$\frac{(ML^2 + ml^2)}{2} \cdot \dot{\theta}^2 + (ml - ML)g \cdot \cos\theta = E_0. \quad (5)$$

Чтобы избавиться в последнем соотношении от неизвестной константы E_0, продифференцируем это соотношение по t. После упрощения полученного выражения приходим к следующему дифференциальному уравнению второго порядка:

$$\ddot{\theta} + \frac{ML - ml}{ML^2 + ml^2} \cdot g \cdot \sin\theta = 0. \quad (6)$$

Обозначим коэффициент при синусе (по условию (1) этот коэффициент положителен) через ω^2:

$$\ddot{\theta} + \omega^2 \sin\theta = 0. \quad (7)$$

Уравнение (7) является нелинейным и в элементарных функциях аналитически не решается. Если предположить, что угол θ мал: $\theta \ll 1$, то тогда мы можем воспользоваться известным приближением $\sin\theta \approx \theta$ и заменить в (7) синус угла θ на сам угол, что приводит нас к линейному однородному уравнению второго порядка:

$$\ddot{\theta} + \omega^2\theta = 0, \quad (8)$$

называемому *уравнением гармонических колебаний*.

Модель • Рассмотрим вывод и решение уравнения (8) с помощью библиотеки `SymPy`. Используя полученное решение, найдем зависимость частоты колебаний метронома от расстояния l крепления его малого груза и построим анимацию колебаний метронома.

1 Подключаем библиотеки `SymPy` и `NumPy`.

[1] Линейная скорость тела, движущегося по окружности, равна произведению угловой скорости на радиус окружности.

2 Создаем символы, необходимые для построения модели, и определяем функцию $\theta(t)$.

```
t, l, L, m, M, g = symbols("t, l, L, m, M, g")
theta = Function('theta')(t)
theta_ = diff(theta, t)
display(theta, theta_)
```

$$\theta(t) \quad \frac{d}{dt}\theta(t)$$

3 Вычисляем полную энергию системы $E = E_{\text{К}} + E_{\text{П}}$ согласно формулам (2) и (3).

```
EP = (m * l - M * L) * g * cos(theta)
EK = (m * l ** 2 + M * L ** 2) / 2 * theta_ ** 2
E = EP + EK
display(E)
```

$$g\left(-LM + lm\right)\cos\left(\theta(t)\right) + \left(\frac{L^2M}{2} + \frac{l^2m}{2}\right)\left(\frac{d}{dt}\theta(t)\right)^2$$

4 Энергия E является постоянной величиной, следовательно, ее производная по t должна равняться нулю. Выполняем указанное дифференцирование и получаем дифференциальное уравнение в форме с неразрешенной второй производной.

```
ode = Eq(diff(E, t), 0)
display(ode)
```

$$-g\left(-LM + lm\right)\sin\left(\theta(t)\right)\frac{d}{dt}\theta(t) + \\ + 2\left(\frac{L^2M}{2} + \frac{l^2m}{2}\right)\frac{d}{dt}\theta(t)\frac{d^2}{dt^2}\theta(t) = 0$$

5 Разрешаем полученное уравнение `ode` относительно второй производной и формируем новое дифференциальное уравнение.

```
sol = solveset(ode, diff(theta, t, t)).args[0]
ode2 = Eq(diff(theta, t, t), sol)
display(ode2)
```

$$\frac{d^2}{dt^2}\theta(t) = -\frac{g\left(LM - lm\right)\sin\left(\theta(t)\right)}{L^2M + l^2m}$$

6 Определяем замену дроби, стоящей перед синусом справа (со знаком минус), на выражение ω^2. Применяем эту замену к уравнению `ode2` и получаем нелинейное уравнение (7).

```
omega = symbols("omega", positive = True)
W = {-ode2.rhs / sin(theta): omega ** 2}
ode3 = ode2.subs(W)
display(ode3)
```

$$\frac{d^2}{dt^2}\theta(t) = -\omega^2 \sin\left(\theta(t)\right)$$

7 Заменяем $\sin\theta$ на θ согласно предположению о малости колебаний и приходим к искомому уравнению гармонических колебаний (8).

```
ode4 = ode3.replace(sin, lambda x: x)
display(ode4)
```

$$\frac{d^2}{dt^2}\theta(t) = -\omega^2\theta(t)$$

8 Решим уравнение `ode4` с учетом начального условия[2] $\theta(0) = \theta_0$, $\dot{\theta}(0) = 0$. Получаем решение в форме чистого косинуса.

[2] Начальное отклонение стержня метронома на заданный угол θ_0 и нулевая начальная скорость.

```
theta0 = symbols("theta0")
ics = {theta.subs(t, 0): theta0,
       diff(theta, t).subs(t, 0): 0}
dsol = dsolve(ode4, theta, ics = ics)
display(dsol)
```

$$\theta(t) = \theta_0 \cos\left(\omega t\right)$$

9 Выполняем обратную замену параметра ω, обращая предварительно словарь `W`, и выражаем найденное решение `dsol` через исходные параметры модели.

```
W2 = {v: k for k, v in W.items()}
dsol2 = dsol.subs(W2)
display(dsol2)
```

$$\theta(t) = \theta_0 \cos\left(t\sqrt{\frac{g\left(LM - lm\right)}{L^2M + l^2m}}\right)$$

10 Построим графики найденного решения для различных значений параметра l (это единственный настраиваемый параметр исходной модели, все остальные параметры фиксированы конструкцией метронома и не могут меняться). Для наглядности покажем каждый график по отдельности, после чего объединим их на общем графике (см. рис. 3).

```
par1 = {theta0: pi / 6, m: 1, M: 10, L: 1, g: 9.8}
p1 = plot(aspect_ratio = (1, 1), show = False)
for l0 in np.arange(0.5, 4, 1):
    par1[l] = l0
    ds = dsol2.rhs.subs(par1)
    p = plot(ds, (t, 0, 6), aspect_ratio = (1, 1))
    p1.extend(p)
p1.show()
```

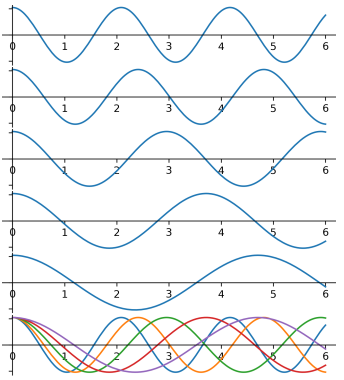


РИС. 3 Графики функции $\theta(t)$ для разных значений параметра l

11 Введенный нами параметр ω называется собственной *циклической* частотой колебаний. Периодом колебаний называется величина $T = 2\pi/\omega$ — это минимальное время, через которое система возвращается в исходное состояние. Обратная периоду величина $\nu = 1/T = \omega/2\pi$ называется частотой колебаний, эта величина измеряется в герцах (число колебаний в секунду). Выразим период колебаний и их частоту через параметры модели.

```
T = (2 * pi / omega).subs(W2)
freq = 1 / T
display(T, freq)
```

$$\frac{2\pi}{\sqrt{\frac{g\,(LM - lm)}{L^2M + l^2m}}}$$

$$\frac{\sqrt{\frac{g\,(LM - lm)}{L^2M + l^2m}}}{2\pi}$$

12 Как следует из графиков на рис. 3, частота колебаний уменьшается с увеличением длины l. Найдем точную зависимость частоты от длины l и построим соответствующие графики для разных значений массы основного груза M. Предварительно выразим из соотношения (1) максимально допустимое значение $l_{\max}$ параметра l, превышение которого должно приводить к переворачиванию маятника. Каждый график будем строить в диапазоне от нуля[3] до $l_{\max}$.

[3] Заметим, что при $l = 0$ метроном превращается в обычный подвесной маятник.

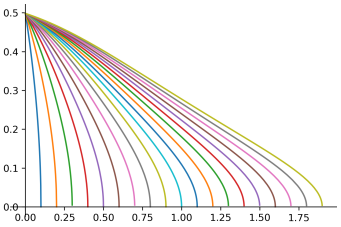


РИС. 4 Зависимость частоты колебаний от длины l для разных значений массы M

```
lmax = M * L / m
par2 = {m: 1, L: 1, g: 9.8}
p2 = plot(show = False)
for M0 in np.arange(0.1, 2, 0.1):
    par2[M] = M0
    f = freq.subs(par2)
    lm = lmax.subs(par2)
    p = plot(f, (l, 0, lm), show = False)
    p2.extend(p)
p2.show()
```

Результат построения приведен на рис. 4. Видно, что зависимость частоты колебаний метронома от расстояния l близка к линейной, что объясняет равномерность шкалы метронома на рис. 1.

13 Переходим к построению анимации колебаний нашего метронома. Определяем параметры модели, подставляем их в решение `dsol` и преобразуем правую часть полученного решения в функцию `Theta`.

```
par = {theta0: pi / 6, m: 1, M: 9,
       L: 0.2, l: 1.2, g: 9.8}
ds = dsol2.subs(par)
Theta = lambdify(t, ds.rhs)
```

14 Вычисляем параметры анимации: `rate` — количество кадров в секунду; `T0` — длительность анимации в секундах (один период колебаний); `frames` — число кадров в анимации; `dt` — длительность одного кадра в секундах.

```
rate = 10
T0 = float(T.subs(par))
frames = int(T0 * rate)
dt = 1 / rate
print(rate, T0, frames, dt)
```

```
10 3.4763817222630955 34 0.1
```

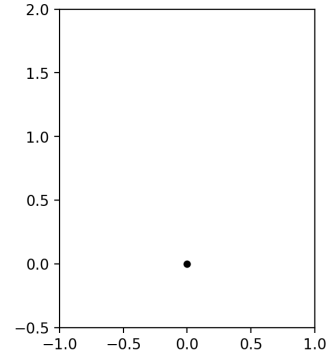


РИС. 5 Шаблон кадра анимации

15 Создаем шаблон одного кадра анимации (рис. 5): подключаем библиотеку `Matplotlib`; создаем поле для кадра (строки 2 и 3); устанавливаем одинаковый масштаб по обеим осям. Далее создаем заготовки элементов, которые будем визуализировать[4]: стержень `S` — отрезок прямой линии, толщиной 6 пикселей, серого цвета; нижний груз `P` — круг радиуса 25 пикселей, красного цвета; верхний груз `Q` — радиус 15 пикселей, синего цвета; точка подвеса `O` — радиус 4 пикселя, черного цвета. Координаты точки подвеса постоянны,

[4] Каждый вызов функции `plot` возвращает кортеж из одного элемента. Запятая после имени переменной, в которую сохраняется созданный объект, нужна для распаковки данного кортежа.

поэтому мы их сразу указываем. Координаты остальных объектов будут вычисляться для каждого кадра заново.

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(xlim = (-1, 1), ylim = (-0.5, 2))
ax.set_aspect('equal')
S, = ax.plot([], [], '-', lw = 6, c = "gray")
P, = plt.plot([], [], 'o', ms = 25, c = 'red')
Q, = plt.plot([], [], 'o', ms = 15, c = 'blue')
O, = plt.plot([0], [0], 'o', ms = 4, c = 'black')
```

16 Вычислим максимально возможную длину $l_{\max}$ для установленных параметров модели. Эту длину будем использовать для прорисовки стержня метронома.

```
lm = float(lmax.subs(par))
print(lm)
```

1.8

17 Определяем функцию `makeframe`, которая должна вычислять координаты всех анимируемых объектов по номеру кадра `i`. Для этого сначала вычисляем значение текущего угла $\theta(t)$, используя функцию `Theta` и учитывая, что время t равно произведению номера текущего кадра `i` на длительность кадра `dt`. Далее находим синус и косинус угла θ. После чего заполняем координаты наших объектов: первый список содержит x-координаты объекта, второй — y-координаты.

```
def makeframe(i):
    th = Theta(i * dt)
    s, c = np.sin(th), np.cos(th)
    S.set_data([-lm * s, par[L] * s],
               [lm * c, -par[L] * c])
    P.set_data(par[L] * s, -par[L] * c)
    Q.set_data(-par[l] * s, par[l] * c)
    return S, P, Q
```

18 Последним действием создадим анимацию с помощью команды `FuncAnimation`, на вход которой передаются: поле кадра, функция прорисовки кадров, количество кадров, длительность одного кадра (в милисекундах). Для показа анимации используем команду `HTML`, которая создает мини-проигрыватель и запускает в нем заданную анимацию.

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
anim = FuncAnimation(fig, makeframe,
                     frames = frames,
                     interval = dt * 1000)
HTML(anim.to_html5_video())
```

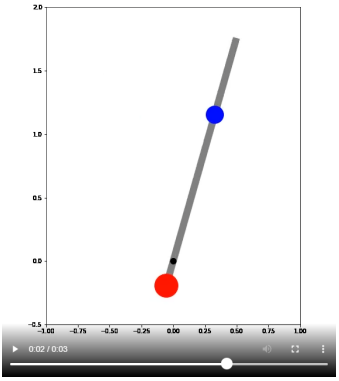


РИС. 6 Кадр построенной анимации

Один кадр созданной анимации показан на рис. 6. Заметим, что по умолчанию показ анимации запускается в бесконечном цикле, а так как мы сделали длительность анимации равной периоду колебаний, то визуально это будет выглядеть как бесконечно работающий метроном.

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Постройте по аналогии с рис. 3 семейство графиков найденного решения $\theta(t)$ для разных значений параметров: θ_0, m, g.

2 Постройте график зависимости частоты колебаний ν от длины l для различных значений ускорения свободного падения, соответствующих разным телам Солнечной системы. Увеличивается или уменьшается частота колебаний при уменьшении g?

3 Постройте графики зависимости периода колебаний T от длины l для разных значений одного из параметров модели (M, m, L, g) при фиксированных знвачениях остальных параметров.

4 Решите начальную задачу для уравнения `ode4` с начальными условиями общего вида:

$$\theta(0) = \theta_0, \ \dot{\theta}(0) = \xi_0. \tag{9}$$

При каких начальных условиях решением начальной задачи будет чистый синус?

5 Общим решением дифференциального уравнения `ode4` является линейная комбинация косинуса и синуса:

```
display(dsolve(ode4, theta))
```

$$\theta(t) = C_1 \sin(\omega t) + C_2 \cos(\omega t)$$

Эту линейную комбинацию можно преобразовать, используя метод вспомогательного аргумента:

$$\theta(t) = A \sin(\omega t + \varphi), \tag{10}$$

где A — амплитуда колебаний:

$$A = \sqrt{C_1^2 + C_2^2}, \tag{11}$$

φ — начальная фаза колебаний, в качестве которой можно взять любое решение системы уравнений

$$\cos\varphi = \frac{C_1}{A}, \ \sin\varphi = \frac{C_2}{A}. \tag{12}$$

Исследуйте зависимость амплитуды колебаний от параметров модели для случая ненулевой начальной угловой скорости $\xi_0 \neq 0$.

6 Колебания подвесного маятника (в том числе и метронома) считаются малыми, если их амплитуда не превосходит 30°. Проанализируйте решение предыдущего упражнения для случая колебаний с амплитудой, большей 30°. Насколько адекватным является найденное нами решение, если амплитуда оказывается больше, чем 180°?

7 Так как решение дифференциального уравнения `ode2` найдено нами в так называемом линейном приближении $\sin\theta \approx \theta$, то следует ожидать, что на этом решении полная энергия системы `E` не будет сохраняться. Чтобы убедиться в этом, постройте график зависимости энергии `E` от времени `t`, подставив в соответствующую формулу вместо функции `theta` решение `dsol2`.

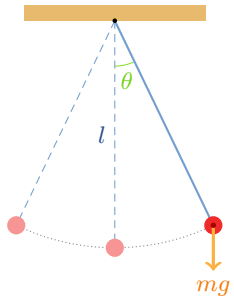


РИС. 7 Простой подвесной маятник

8 Рассмотрите по аналогии с моделью метронома модель простого подвесного маятника (рис. 7), решите полученное дифференциальное уравнение в случае малых колебаний, получите формулы частоты, периода и амплитуды колебаний, постройте анимацию движения маятника для какого-нибудь одного заданного набора значений параметров модели.

9 Решите с помощью `SymPy` следующие начальные задачи для линейных однородных уравнений с постоянными коэффициентами:

1) $y'' = -4y$, $y(0) = 0$, $y'(0) = 2$;
2) $y'' = -y$, $y(0) = -2$, $y'(0) = 0$;
3) $y'' = -2y$, $y(0) = 1$, $y'(0) = -1$;
4) $y'' = -a^2 y$, $y(0) = 3$, $y'(0) = 1$.

10 Постройте анимацию найденного в главе решения $\theta(t)$ по одному из следующих параметров модели: l, θ_0, m, g. Каждый кадр анимации должен соответствовать своему значению выбранного параметра (в некотором диапазоне его изменения) при фиксированных значениях остальных параметров.

11 Постройте анимации следующих графиков (указаны уравнение кривой, параметр анимации, пределы рисования графика):

1) $x^2 + a$, $a \in [-3, 3]$, $x \in [-3, 3]$, $y \in [-3, 3]$;
2) $(x - a)^2$, $a \in [-3, 3]$, $x \in [-3, 3]$, $y \in [-3, 3]$;
3) $\cos(x - a)$, $a \in [0, 2\pi]$, $x \in [-2\pi, 2\pi]$, $y \in [-1, 1]$;
4) $\sin(a)\cos(x)$, $a \in [0, 2\pi]$, $x \in [-\pi, \pi]$, $y \in [-1, 1]$.

12 Нелинейное уравнение (7) в общем виде (без начального условия) аналитически не решается. Но его можно решить численно, с помощью одного из приближенных методов, например с помощью метода Эйлера. Для этого предварительно надо преобразовать данное уравнение второго порядка к системе двух уравнений первого порядка введением дополнительной неизвестной функции — угловой скорости $\xi(t)$:

$$\frac{d\theta}{dt} = \xi, \quad \frac{d\xi}{dt} = -\omega^2 \sin\theta. \tag{13}$$

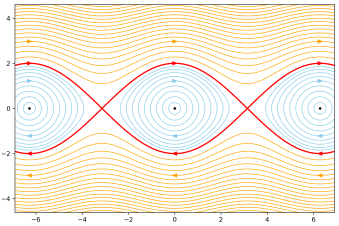


РИС. 8 Фазовый портрет уравнения (13), по горизонтали отложен угол θ, по вертикали — угловая скорость $\xi = \dot{\theta}$

Постройте с помощью такого приближенного решения графики зависимости периода колебаний метронома от величины начального отклонения θ_0 (при нулевой начальной угловой скорости), сравните эти графики с соответствующими графиками из упражнения 3.

13 Перепишем соотношение (5), из которого была выведена система (13), с использованием введенного обозначения угловой скорости $\xi = \dot{\theta}$ и полученного ранее выражения для собственной частоты ω:

$$\xi^2 - 2\omega^2 \cos\theta = C, \tag{14}$$

где C — константа, зависящая от параметров модели и начального условия (начальной энергии E_0). То есть в любой момент времени угол θ и угловая скорость ξ оказываются связанными формулой (14). Соответствующее этой формуле семейство кривых на плоскости $O\theta\xi$ (см. рис. 8) называется *фазовым портретом* системы (13), сама плоскость называется фазовой, а каждая отдельная кривая — фазовой траекторией. Выполните указанное построение фазового портрета с помощью SymPy.

ГЛАВА 13
Пружинный маятник

Задача • Рассмотрим простую модель пружинного маятника, представляющего собой подвешенный с помощью пружины груз массы m (см. рис. 1). В такой модели на груз маятника действуют как минимум две силы — сила тяжести F_1, направленная вниз, и сила упругости F_2 со стороны сжатой или растянутой пружины. Величина силы упругости определяется законом Гука[1]: *величина силы упругости пружины пропорциональна изменению длины этой пружины (относительно ее естественной длины)*. Коэффициент пропорциональности k в законе Гука называется *жесткостью* пружины.

Также будем предполагать, что на груз маятника, кроме двух указанных выше сил, могут действовать еще две дополнительные силы. Первая из них — это сила сопротивления F_3 со стороны среды, в которую помещен груз маятника. Если в качестве среды выступает какая-нибудь жидкость, например вода или масло, то сила сопротивления оказывается пропорциональной скорости груза v и направленной против этой скорости[2]. Второй дополнительной силой может быть внешняя (то есть не зависящая ни от положения, ни от скорости груза) периодическая сила F_4, величина которой определяется формулой:

$$F_4 = f_0 \sin \omega_0 t, \qquad (1)$$

где f_0 — амплитуда данной силы, а ω_0 — ее циклическая частота.

Введем одномерную систему координат (рис. 1), ось Ox которой направим вертикально вверх, а начало координат O поместим в точку, соответствующую недеформированному состоянию пружины (т. е.

[1] Роберт Гук — английский естествоиспытатель и изобретатель XVII века. В настоящее время не существует ни одного подтвержденного его портрета.

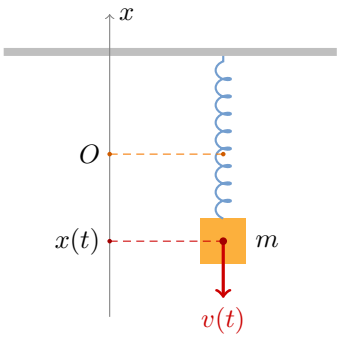


РИС. 1 Простейший пружинный маятник

[2] Если средой является воздух, то сила сопротивления пропорциональна *квадрату* скорости, что на самом деле существенно усложняет рассматриваемую модель.

в точке O сила упругости равна нулю). Функция $x(t)$ определяет положение груза в момент времени t. Тогда скорость груза равна $v = \dot{x}$. Таким образом, сила тяжести равна $F_1 = -mg$, сила упругости $F_2 = -kx$, а сила сопротивления $F_3 = -rv = -r\dot{x}$, где r — коэффициент пропорциональности, определяемый свойствами среды и геометрией груза маятника.

Записываем второй закон Ньютона применительно к нашей системе:

$$m\ddot{x} = -mg - kx - r\dot{x} + f_0 \sin \omega_0 t, \tag{2}$$

где вторая производная $\ddot{x}$, как обычно, соответствует ускорению груза. Перегруппируем слагаемые и поделим все уравнение на m:

$$\ddot{x} + \frac{r}{m}\dot{x} + \frac{k}{m}x = -g + \frac{f_0}{m} \sin \omega_0 t. \tag{3}$$

Полученное соотношение является линейным неоднородным дифференциальным уравнением второго порядка с постоянными коээфициентами. Для его однозначного разрешения требуются два дополнительных условия. Будем предполагать, что начальное отклонение груза равно x_0, а его начальная скорость равна нулю:

$$x(0) = x_0, \ \dot{x}(0) = 0. \tag{4}$$

Модель • Построим описанную модель средствами библиотеки `SymPy`. Рассмотрим три важных частных случая этой модели: свободные колебания ($r = 0$, $f_0 = 0$), затухающие колебания ($r \neq 0$, $f_0 = 0$) и вынужденные колебания ($r = 0$, $f_0 \neq 0$), приводящие, в частности, к явлению резонанса.

1 Подключаем библиотеки `SymPy`, `NumPy` и `Matplotlib`.

2 Создаем символы `m`, `k`, `r`, `f0`, `omega0`, `g` для всех параметров, входящих в уравнение (2), указывая, что все они являются неотрицательными величинами. Определяем символ t и вводим функцию $x(t)$.

3 Задаем дифференциальное уравнение (2) в максимально общем виде, т. е. с учетом всех четырех рассмотренных выше сил.

```
ode = Eq(m * diff(x, t, t), -m * g - k * x -
         r * diff(x, t) + f0 * sin(omega0 * t))
display(ode)
```

$$m\frac{d^2}{dt^2}x(t) = f_0\sin(\omega_0 t) - gm - kx(t) - r\frac{d}{dt}x(t)$$

4 Определяем символ `x0` и задаем начальные условия (4).

```
x0 = symbols("x0")
ic = {x.subs(t, 0): x0, diff(x, t).subs(t, 0): 0}
```

5 Рассмотрим первый частный случай свободных колебаний, когда на груз действуют только сила тяжести и сила упругости. Для этого обнулим параметры r и f_0 в дифференциальном уравнении и решим его.

```
par1 = {r: 0, f0: 0}
dsol1 = dsolve(ode.subs(par1), x, ics = ic)
display(dsol1)
```

$$x(t) = -\frac{gm}{k} + \frac{(gm+kx_0)\cos\left(\frac{\sqrt{k}t}{\sqrt{m}}\right)}{k}$$

6 Видно, что найденная зависимость $x(t)$ является косинусоидальной, при этом колебания происходят относительно величины $-gm/k$. Сохраним эту величину в переменной `x1`.

```
x1 = dsol1.rhs.replace(cos, lambda x: 0)
display(x1)
```

$$-\frac{gm}{k}$$

7 Построим график семейства интегральных кривых найденного решения для различных значений параметра k при фиксированных значениях остальных параметров модели для начального условия $x_0 = 0$.

```
p1 = plot(size = (8, 4), show = False)
for k_ in np.arange(2.0, 8.0, 0.5):
    ds = dsol1.rhs.subs({m: 1, k: k_, x0: 0,
                         g: 9.8})
    p = plot(ds, (t, 0, 2 * pi), show = False)
    p1.extend(p)
p1.show()
```

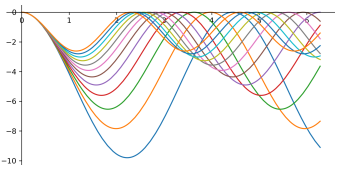


РИС. 2 Свободные колебания пружинного маятника для различных значений параметра k

Из построенного графика (см. рис. 2) видно, что чем меньше жесткость пружины, тем большей является амплитуда колебаний и тем ниже располагается точка равновесия системы, относительно которой происходят ее колебания.

8 Для сравнения построим график того же семейства кривых для разных значений параметра x_0, который явно показывает, что колебания являются гармоническими (рис. 3).

```
p2 = plot(size = (8, 4), show = False)
for x0_ in np.arange(-7.0, 2.1, 0.5):
    ds = dsol1.rhs.subs({m: 1, k: 4, x0: x0_,
                         g: 9.8})
    p = plot(ds, (t, 0, 2 * pi), show = False)
    p2.extend(p)
p2.show()
```

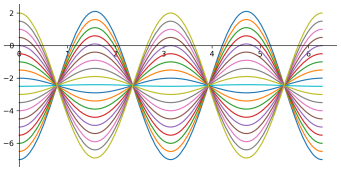


РИС. 3 Свободные колебания пружинного маятника для различных значений параметра x_0

9 Перейдем теперь к случаю затухающих колебаний. Для этого нужно в уравнении `ode` обнулить только коэффициент `f0`. Так как решение получающегося после такой замены уравнения в общем случае оказывается очень громоздким, сразу подставим в уравнение числовые значения параметров `k` и `m`, а в найденное решение — ноль вместо `x0`. Получаем решение, зависящее от параметров `g` и `r`.

```
par2 = {m: 1, k: 4, f0: 0}
dsol2 = dsolve(ode.subs(par2), x, ics = ic)
dsol2 = dsol2.subs(x0, 0)
display(dsol2)
```

$$x(t) = -\frac{g}{4} + \left(-\frac{gr}{8\sqrt{r-4}\sqrt{r+4}} + \frac{g}{8}\right) e^{\frac{t\left(-r-\sqrt{r-4}\sqrt{r+4}\right)}{2}} + \\ + \left(\frac{gr}{8\sqrt{r-4}\sqrt{r+4}} + \frac{g}{8}\right) e^{\frac{t\left(-r+\sqrt{r-4}\sqrt{r+4}\right)}{2}}$$

10 Известно, что имеется два основных режима затухающих колебаний. В подкритическом режиме, когда сила сопротивления (т. е. параметр r) относительно невелика, колебания маятника происходят, но их амплитуда постепенно уменьшается до нуля. В надкритическом режиме (при большой силе сопротивления) колебания как таковые не происходят, маятник просто возвращается в положение своего равновесия. Для пружинного маятника критическим значением, разделяющим эти два режима, является $r_0 = 2\sqrt{km}$. Для выбранного нами в предыдущем пункте набора параметров $r_0 = 4$. При $r < 4$ в решении `dsol2` подкоренные выражения в показателях экспонент оказываются отрицательными, т. е. сами показатели — комплексными. Это значит, что после применения к данным экспонентам формулы Эйлера в решении появятся синусы

и косинусы. Так как SymPy умеет работать с комплексными числами, то мы можем построить графики решения dsol2 даже для подкритических значений параметра r. Подкритические кривые выделим оранжевым цветом, надкритические — голубым цветом.

```
p3 = plot(size = (8, 6), show = False)
for r_ in np.arange(0.5, 8.0, 0.5):
    ds = dsol2.rhs.subs({r: r_, x0: 0, g: 9.8})
    c = "skyblue" if r_ < 4 else "orange"
    p = plot(ds, (t, 0, 2 * pi), line_color = c,
             show = False)
    p3.extend(p)
X1 = x1.subs(par2).subs(g, 9.8)
p = plot(X1, (t, 0, 2 * pi), line_color = "black",
         show = False)
p3.extend(p)
p3.show()
```

Результат построения показан на рис. 4. Хорошо видны затухающие колебания в подкритическом режиме. В надкритическом режиме кривые являются монотонно убывающими к положению равновесия $x = x_1$ (прямая черного цвета).

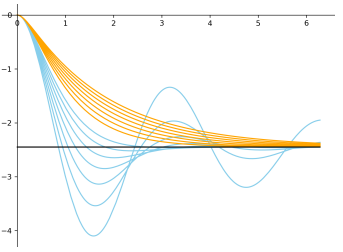


РИС. 4 Затухающие колебания пружинного маятника

11 Из графика 4 видно, что между оранжевыми и голубыми кривыми есть зазор, в котором должно располагаться так называемое критическое решение для случая $r = r_0 = 4$. Прямая подстановка этого значения в решение dsol2 приводит к делению на ноль, поэтому SymPy игнорирует построение данной кривой. Чтобы построить недостающую кривую, найдем критическое решение в явном виде.

```
par3 = {m: 1, k: 4, r: 4, f0: 0}
dsol3 = dsolve(ode.subs(par3), x, ics = ic)
display(dsol3)
```

$$x(t) = -\frac{g}{4} + \left(\frac{g}{4} + t\left(\frac{g}{2} + 2x_0\right) + x_0\right)e^{-2t}$$

12 Добавим найденное критическое решение dsol3 к построенному ранее графику p3, выделив это решение красным цветом (см. рис. 5).

```
p4 = plot(size = (8, 6), show = False)
p4.extend(p3)
ds = dsol3.rhs.subs({x0: 0, g: 9.8})
p = plot(ds, (t, 0, 2 * pi), line_color = "red",
         show = False)
p4.extend(p)
p4.show()
```

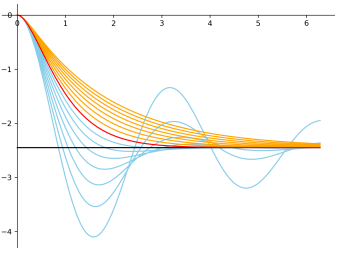


РИС. 5 Критический режим (красная кривая) затухающих колебаний пружинного маятника

13 Последний случай, который мы должны рассмотреть, — это случай вынужденных колебаний под действием внешней периодической силы. В этот раз мы должны обнулить параметр `r` в уравнении `ode`. Хотя в общем виде такое уравнение и решается библиотекой `SymPy`, для большей наглядности мы сразу подставим в уравнение параметры $m = 1$, $k = 4$, $f_0 = 1$.

```
par5 = {m: 1, k: 4, r: 0, f0: 1}
dsol5 = dsolve(ode.subs(par5), x, ics = ic)
display(dsol5)
```

$$x(t) = -\frac{g}{4} + \frac{\omega_0 \sin(2t)}{2\omega_0^2 - 8} + \left(\frac{g}{4} + x_0\right)\cos(2t) - \frac{\sin(\omega_0 t)}{\omega_0^2 - 4}$$

В полученном решении первое слагаемое соответствует точке покоя x_1, второе и третье — колебаниям маятника с его собственной частотой ω, а последнее слагаемое — колебаниям с частотой внешней силы ω_0.

14 Построим графики найденного решения для разных значений частоты ω_0. В качестве начального условия `x0` выберем точку покоя маятника `X1`[3].

[3] Вычисленную нами при тех же параметрах модели в пункте 10.

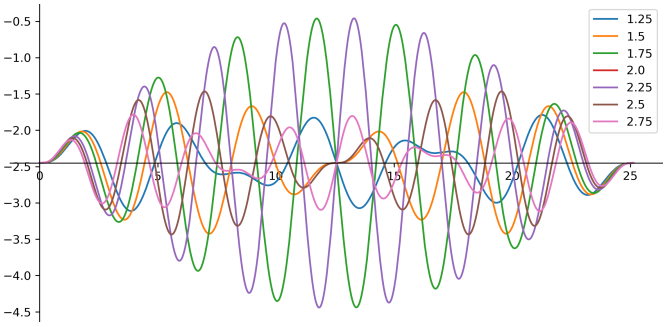


РИС. 6 Вынужденные колебания пружинного маятника

```
p5 = plot(size = (8, 4), show = False)
for om in np.arange(1.25, 3, 0.25):
    ds = dsol5.rhs.subs({omega0: om, x0: X1, g: 9.8})
    p = plot(ds, (t, 0, 8 * pi), show = False,
             label = om)
    p5.extend(p)
p5.legend = True
p5.show()
```

Результат построения приведен на рис. 6. Видно, что колебания стали более сложными, их амплитуда меняется со временем и зависит от частоты ω_0 внешней

силы. Известно, что критическим значением частоты ω_0 является собственная частота ω свободных колебаний того же маятника. В нашем случае[4] критическая частота равна 2. При приближении параметра ω_0 к этому значению амплитуда колебаний начинает возрастать, такое явление называется *резонансом*.

[4] См. знаменатель во втором и четвертом слагаемых в решении `dsol5`.

15 При точном совпадении двух указанных частот ($\omega_0 = \omega$) решение нашего дифференциального уравнения меняется кардинально. На рис. 6 это решение отсутствует, т. к. подстановка $\omega_0 = 2$ в решение `dsol5` приводит к делению на ноль. Найдем резонансное решение явным образом.

```
par6 = {m: 1, k: 4, r: 0, f0: 1, omega0: 2}
dsol6 = dsolve(ode.subs(par6), x, ics = ic)
display(dsol6)
```

$$x(t) = -\frac{g}{4} + \left(\frac{g}{4} - \frac{t}{4} + x_0\right)\cos(2t) + \frac{\sin(2t)}{8}$$

Коэффициент перед косинусом линейно зависит от t, что и приводит к неограниченному росту амплитуды колебаний при $t \to \infty$ — это явление называется *математическим* резонансом.

16 Последним действием добавим к общему графику `p5` найденное резонансное решение `dsol6`, выделив его красным цветом (см. рис. 7).

```
p6 = plot(size = (8, 6), show = False)
p6.extend(p5)
ds = dsol6.rhs.subs({x0: X1, g: 9.8})
p = plot(ds, (t, 0, 8 * pi), line_color = "red",
         show = False)
p6.extend(p)
p6.show()
```

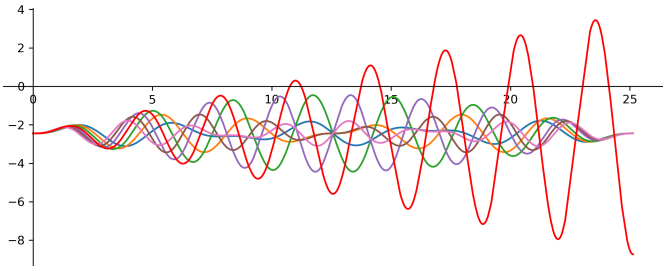


РИС. 7 Явление математического резонанса (красная кривая)

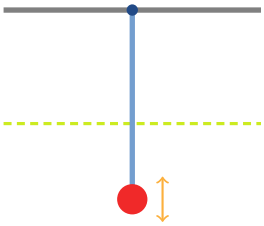


РИС. 8 Схема одного кадра анимации

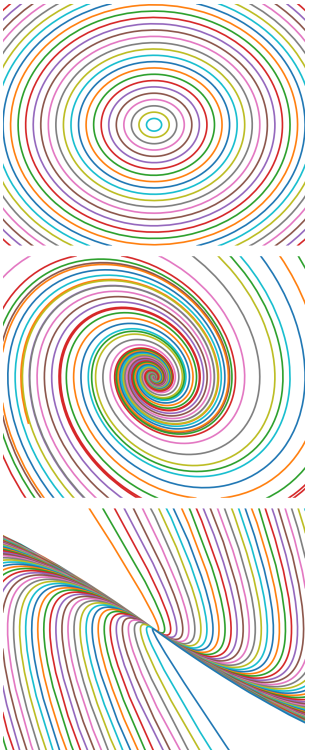


РИС. 9 Фазовые портреты: свободные колебания, затухающие подкритические и надкритические колебания

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Постройте по аналогии с моделью из предыдущей главы анимацию поведения пружинного маятника, описываемого дифференциальным уравнением `ode` с заданным набором параметров и заданным начальным условием. Каждый кадр такой анимации должен содержать пять объектов (см. рис. 8): горизонтальную линию, представляющую поверхность, к которой прикреплена пружина; горизонтальную пунктирную линию, показывающую положение равновесия маятника; точку подвеса пружины; груз маятника; пружину, которую можно изобразить простой линией. Переменными вертикальными координатами обладают только груз маятника и нижний конец пружины. Протестируйте анимацию на колебаниях различных типов.

2 Постройте по аналогии с рис. 2 графики решения `dsol1` (случай свободных колебаний) для разных значений параметров g и m. Как влияет на частоту и амплитуду колебаний увеличение или уменьшение этих двух параметров?

3 Постройте и визуализируйте решение всех трех рассмотренных в главе случаев (свободные колебания, затухающие колебания, вынужденные колебания) для начального условия $x(0) = 0$, $\dot{x}(0) = v_0$.

4 Рассмотрите общий случай, когда на груз маятника действуют все четыре силы. Интересной особенностью такой модели является то, что в ней оказывается невозможным явление математического резонанса, т. е. амплитуда колебаний при наличии силы сопротивления не может возрастать неограниченно.

5 Постройте анимированный вариант графика на рис. 6, в котором параметром анимации является частота внешней силы ω_0.

6 Постройте фазовые портреты свободных и затухающих колебаний. Для этого надо взять решение $x(t)$ соответствующего дифференциального уравнения, вычислить скорость $v(t) = \dot{x}(t)$ и построить семейство графиков $v(x)$ для различных значений x_0 в параметрической форме, используя в качестве параметра время t.

7 Напишите функцию, которая принимает на вход линейное однородное дифференциальное уравнение с постоянными коэффициентами относительно функции $y(x)$

$$y^{(n)} + a_{n-1}y^{(n-1)} + \cdots + a_1 y' + a_0 y = 0 \qquad (5)$$

и составляет соответствующее ему так называемое *характеристическое уравнение* относительно параметра λ

$$\lambda^n + a_{n-1}\lambda^{n-1} + \cdots + a_1\lambda + a_0 = 0. \qquad (6)$$

Корни уравнения (6) называются собственными значениями дифференциального уравнения (5) и полностью определяют его общее решение. Для построения характеристического уравнения нужно выполнить в дифференциальном уравнении замену $x(t) = e^{\lambda t}$, вычислить все производные и сократить полученное соотношение на $e^{\lambda t}$.

8 Решите следующие линейные однородные уравнения:

1) $y''' - 2y'' = 0$;
2) $y''' + 8y = 0$;
3) $y^{(4)} - y = 0$;
4) $y^{(4)} + 5y'' + 4y = 0$.

9 Решите следующие начальные задачи для линейных неоднородных уравнений:

1) $y'' - 2y' + y = x^2 - 1$, $y(0) = 1$, $y'(0) = -1$;
2) $y'' + 3y' + 2y = 5\sin x$, $y(0) = -2$, $y'(0) = 0$;
3) $y'' + 4y = e^{-2x}$, $y(0) = -1$, $y'(0) = 2$;
4) $y'' - 3y' = 4e^{3x}$, $y(0) = 0$, $y'(0) = 1$.

10 Постройте модель движения груза, соединенного пружинами с заданной жесткостью с двумя неподвижными стенками (рис. 10). Решите и визуализируйте полученное дифференциальное уравнение. Трением в системе пренебречь.

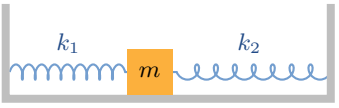


РИС. 10 Маятник с двумя пружинами

11 Постройте модель движения двух грузов с массами m_1 и m_2, соединенных пружиной с жесткостью k (рис. 11). Неизвестными в такой модели должны быть две функции $x_1(t)$ и $x_2(t)$, определяющие положение грузов в момент времени t. Задайте начальные положения и скорости грузов, решите построенное дифференциальное уравнение, визуализируйте найденное решение. Трением в системе пренебречь.

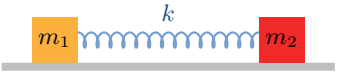


РИС. 11 Модель с двумя грузами

12 Определение собственных частот колебательной системы является важной практической задачей. Если система описывается линейным дифференциальным уравнением высокого порядка, то аналитическое нахождение собственных частот оказывается невозможным. Однако собственные частоты системы, заданной линейным однородным дифференциальным уравнением, можно найти численно с помощью какого-нибудь приближенного метода решения дифференциальных уравнений, например с помощью метода Эйлера. Идея подхода основана на моделировании явления резонанса. Рассмотрим для примера простейшую колебательную систему, описываемую дифференциальным уравнением второго порядка

$$\ddot{x} + 4x = 0, \tag{7}$$

собственная частота колебаний которой равна $\omega = 2$. Добавим к системе внешнюю периодическую силу, действующую с частотой ω_0:

$$\ddot{x} + 4x = \sin \omega_0 t, \tag{8}$$

и преобразуем уравнение в систему:

$$\dot{x} = v, \ \dot{v} = -4x + \sin \omega_0 t. \tag{9}$$

Теперь будем решать полученную систему численно с нулевыми начальными условиями для разных значений ω_0 на одном и том же интервале времени $t \in [0, T]$. Для каждого найденного решения запомним его максимальную (по модулю) амплитуду, после чего построим график полученной зависимости максимальной амплитуды колебаний от частоты ω_0. Максимумы на этом графике и будут соответствовать приближенным значениям собственных частот рассматриваемой системы. Например, для системы (7) описанная схема дает график, показанный на рис. (12), из которого хорошо видно, что пик приходится как раз на собственную частоту $\omega = 2$.

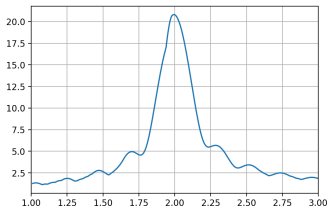


РИС. 12 Зависимость максимальной амплитуды от частоты ω_0

13 Примените подход, описанный в предыдущем упражнении, для численного определения собственных частот двойного пружинного маятника, показанного на рис. 13.

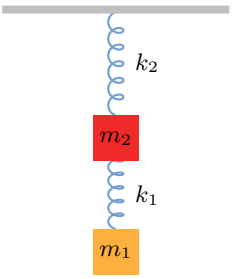


РИС. 13 Двойной пружинный маятник

ГЛАВА 14
Модель Лотки—Вольтерры

Задача • В 1925 году Альфред Лотка и на год позже в 1926 году Вито Вольтерра независимо друг от друга предложили простую математическую модель взаимодействия двух видов, в которой один вид (жертвы) служит источником пищи для второго вида (хищники).

Пусть $x(t)$ обозначает размер популяции хищников в момент времени t, а $y(t)$ — размер популяции жертв. Предполагается, что каждая популяция в отсутствие другой популяции подчиняется закону естественного роста[1], причем в популяции жертв коэффициент роста (удельная скорость роста популяции) положителен, т. е. популяция жертв неограниченно растет, а в популяции хищников этот коэффициент отрицателен, т. е. в отсутствие жертв хищники вымирают:

$$\dot{x} = -\alpha x,\ \dot{y} = \gamma y,\ \alpha, \gamma > 0. \tag{1}$$

Взаимодействие популяций (т. е. поедание жертв хищниками) ведет к уменьшению числа жертв и увеличению числа хищников. В простейшем случае изменение размера каждой популяции будет пропорциональным частоте встреч двух видов, т. е. произведению xy размеров их популяций. Добавляя это взаимодействие к уравнениям (1), мы приходим к системе дифференциальных уравнений

$$\begin{aligned} \dot{x} &= -\alpha x + \beta xy, \\ \dot{y} &= \gamma y - \delta xy, \end{aligned} \tag{2}$$

называемой моделью Лотки—Вольтерры, или моделью «хищник—жертва». Все параметры модели α, β, γ и δ являются строго положительными.

Альфред Джеймс Лотка

Вито Вольтерра

[1] См. главу 6.

Модель • Система уравнений (2) является нелинейной, и известно, что она не решается аналитически. Поэтому мы рассмотрим несколько подходов к приближенному, в том числе к графическому, решению этой системы с помощью библиотек SymPy и Matplotlib.

1 Подключаем библиотеки SymPy, NumPy и Matplotlib.

2 Создаем символы для положительных параметров модели α, β, γ и δ. Так как мы начнем с приближенного решения системы (2), то функции $x(t)$ и $y(t)$ объявим пока тоже как символы, а не как функции.

```
al, bt, gm, dl = symbols("alpha beta gamma delta",
                         positive = True)
x, y = symbols("x y")
```

3 Создадим и запомним в переменных dxdt и dydt выражения, стоящие в правых частях системы (2).

```
dxdt = -al * x + bt * x * y
dydt = gm * y - dl * x * y
display(dxdt, dydt)
```

$$-\alpha x + \beta xy$$
$$-\delta xy + \gamma y$$

4 Для приближенного решения системы (2) воспользуемся командой odeint библиотеки SciPy. Для этого предварительно преобразуем выражения dxdt и dydt в функции Python.

```
fx = lambdify([x, y, al, bt], dxdt)
fy = lambdify([x, y, gm, dl], dydt)
```

5 Определим функцию dsdt, которая принимает на вход вектор s, составленный из двух переменных x и y, переменную t, от которой зависят x и y[2], и вспомогательные параметры α, β, γ и δ, а возвращает вектор из двух производных $\dot{x}$ и $\dot{y}$, вычисленных согласно системе (2). В первой строке этой функции мы распаковываем вектор в скаляры, а во второй — вычисляем правые части системы.

```
def dsdt(s, t, a, b, g, d):
    x, y = s
    return [fx(x, y, a, b), fy(x, y, g, d)]
```

[2] В нашем случае правая часть системы не зависит явно от t, но эту переменную в списке аргументов надо все-таки указывать для совместимости с общим случаем.

6 Подключаем команду `odeint`. Фиксируем параметры модели: $\alpha = \beta = 0.5$, $\gamma = \delta = 1$. Определяем сетку `T` по переменной t, на которой будем искать приближенное решение заданной системы дифференциальных уравнений. Наконец, вызываем команду `odeint`. Первым ее аргументом идет функция, вычисляющая правые части всех уравнений системы. Опциональный аргумент `y0` задает начальное условие $x(0) = 0.25$, $y(0) = 1$. Аргумент `t` задает сетку, на которой будут выполняться вычисления. В аргументе `args` указываются в форме кортежа все дополнительные параметры, которые будут переданы в функцию `dsdt` (см. определение этой функции в пункте 5).

```
from scipy.integrate import odeint
par = (0.5, 0.5, 1.0, 1.0)
T = np.linspace(0, 50, 1000)
dsol = odeint(dsdt, y0 = [0.25, 1.0],
              t = T, args = par)
print(dsol)
```

```
[[0.25       1.        ]
 [0.25011893 1.0382489 ]
 [0.25048208 1.07794783]
 ...
 [0.37753404 0.27373435]
 [0.37077453 0.28244464]
 [0.36421712 0.29152923]]
```

7 Результатом работы команды `odeint`, как видно, является двумерный массив, столбцы которого соответствуют неизвестным функциям, а строки — переменной, в нашем случае времени t. Построим графики найденного решения, соответствующие двум столбцам массива `dsol`. Команда в первой строке создает рисунок (figure) и график (axes), команды во второй и третьей строках строят на графике две кривые $x(t)$ и $y(t)$, в четвертой строке добавляем к графику легенду на основе указанных меток кривых.

```
fig, ax = plt.subplots()
ax.plot(T, dsol[:, 0], label = "хищники")
ax.plot(T, dsol[:, 1], label = "жертвы")
ax.legend()
plt.show()
```

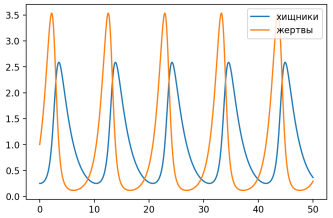


РИС. 1 Приближенное решение системы (2)

Результат построения показан на рис. 1. Очевидным является колебательный характер этих кривых.

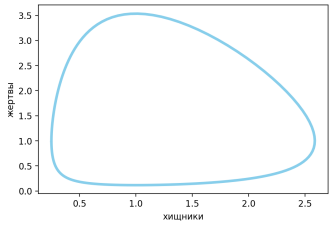


РИС. 2 Фазовая траектория, соответствующая решению на рис. 1

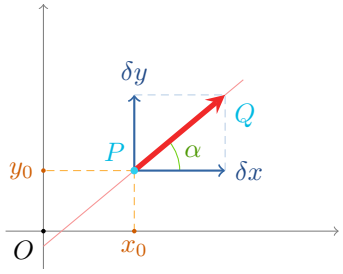


РИС. 3 Построение указателя в точке P

[3] Построение большого числа указателей — это весьма трудоемкая операция, поэтому ее выполнение обычно доверяют компьютерам.

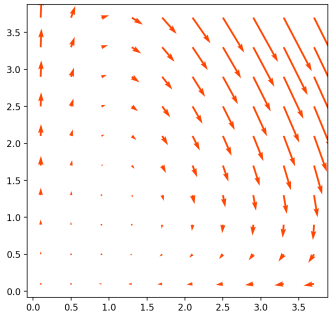


РИС. 4 Поле направлений системы (2)

8 Используя те же данные, можем построить фазовую траекторию на плоскости Oxy.

```
fig, ax = plt.subplots()
ax.plot(dsol[:, 0], dsol[:, 1],
        "skyblue", lw = 3)
ax.set_xlabel('хищники')
ax.set_ylabel('жертвы')
plt.show()
```

Траектория оказалась замкнутой, что также согласуется с колебательным поведением решения на рис. 1.

9 Приближенным и очень простым методом построения фазового портрета любой автономной системы второго порядка, например системы (2), является построение *поля направлений* этой системы. Выберем на фазовой плоскости произвольную точку $P = (x_0, y_0)$ (рис. 3) и подставим ее координаты в правые части системы (2), это даст нам значения производных

$$\delta x = \dot{x}(x_0, y_0) \text{ и } \delta y = \dot{y}(x_0, y_0) \qquad (3)$$

в точке P. Построим теперь направленный отрезок, называемый *указателем*, который соединяет точку P с точкой $Q(x_0 + \delta x, y0 + \delta y)$. Так как любой указатель является фактически отрезком касательной к фазовой траектории в заданной точке, то, применив описанную процедуру для большего числа точек, равномерно распределенных по некоторой области фазовой плоскости, мы получим приближенный фазовый портрет исходной системы дифференциальных уравнений в данной области[3]. Так как длины указателей могут варьироваться в широком диапазоне, то их, как правило, некоторым способом нормируют, например можно просто использовать указатели фиксированной длины. В библиотеке `Matplotlib` имеется специальная команда `quiver` для построения векторных полей, воспользуемся ею для построения поля направлений системы (2). Сначала с помощью команды `meshgrid` задаем сетку, по которой будут строиться указатели. Затем вычисляем в каждой точке сетки значения производных, используя функции `fx` и `fy`. В седьмой строке строим искомое поле направлений: первые два аргумента команды `quiver` задают сетку, следующие два — координаты векторов. Результат построения показан на рис. 4.

```
fig, ax = plt.subplots(figsize = (6, 6))
xrange = np.arange(0.1, 4.05, 0.4)
yrange = np.arange(0.1, 4.05, 0.4)
X, Y = np.meshgrid(xrange, yrange)
U = fx(X, Y, 0.5, 0.5)
V = fy(X, Y, 1.0, 1.0)
ax.quiver(X, Y, U, V, color = "orangered")
plt.show()
```

10 Команда `quiver` автоматически выполняет нормировку длин указателей к некоторому среднему значению. В нашем случае разброс длин очень большой, поэтому часть указателей фактически превратились в точки. Выполним свою нормировку (строки 2 и 3), поделив каждый элемент массивов `U` и `V`, т. е. координат указателей, на длину соответствующего указателя. Результат построения приведен на рис. 5.

```
fig, ax = plt.subplots(figsize = (6, 6))
U1 = U / np.sqrt(U ** 2 + V ** 2)
V1 = V / np.sqrt(U ** 2 + V ** 2)
ax.quiver(X, Y, U1, V1, color = "orangered")
plt.show()
```

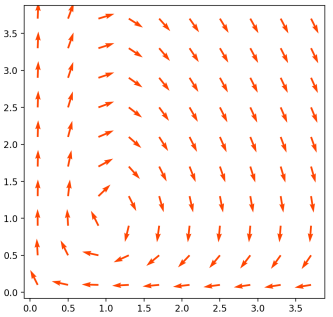


РИС. 5 Поле направлений системы (2) с указателями равной длины

11 Совместим (рис. 6) построенное поле направлений с вычисленной ранее в пункте 8 фазовой траекторией `dsol`.

```
fig, ax = plt.subplots(figsize = (6, 6))
ax.quiver(X, Y, U1, V1, color = "orangered")
ax.plot(dsol.T[0][:216], dsol.T[1][:216],
        "skyblue", lw = 3)
plt.show()
```

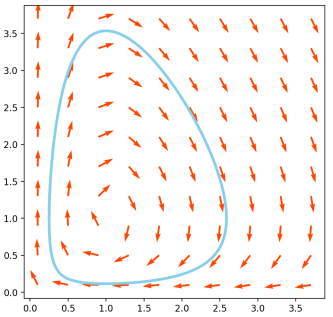


РИС. 6 Поле направлений и фазовая траектория системы (2)

12 Хотя сама система (2) относительно неизвестных функций $x(t)$ и $y(t)$ и не решается аналитически, дифференциальное уравнение для ее фазовых траекторий может быть решено аналитически в неявном виде. Поделим второе уравнение системы (2) на первое и заменим отношение $\dot{y}/\dot{x}$ на производную dy/dx:

$$\frac{dy}{dx} = \frac{y(\gamma - \delta x)}{x(\beta y - \alpha)}. \tag{4}$$

Получили дифференциальное уравнение первого порядка с разделяющимися переменными относительно неизвестной функции $y(x)$. Решим это уравнение с помощью `SymPy`. Для этого введем новый символ `yf` для функции $y(x)$, так как в данный момент символ `y` является символом переменной. Составляем дифференциальное уравнение (4), используя выражения `dxdt` и `dydt` и выполняя замену `y` на `yf`.

```
yf = Function("y")(x)
ode = Eq(diff(yf, x), (dydt / dxdt).subs(y, yf))
display(ode)
plt.show()
```

$$\frac{d}{dx}y(x) = \frac{-\delta x y(x) + \gamma y(x)}{-\alpha x + \beta x y(x)}$$

13 Решаем уравнение `ode`, как уравнение с разделяющимися переменными, без упрощения результата, т. е. в неявной форме.

```
dsol2 = dsolve(ode, yf, hint = "separable",
               simplify = False)
display(dsol2)
```

$$-\alpha \log\left(y(x)\right) + \beta y(x) = C_1 - \delta x + \gamma \log\left(x\right)$$

14 Найденное решение является уравнением фазовых траекторий исходной системы. Для построения этих траекторий, в принципе, можно использовать команду `plot_implicit` библиотеки `SymPy`, но более качественно и быстрее это делается с помощью команды `contour` библиотеки `Matplotlib`. Для этого выразим из найденного решения константу C_1 и запомним полученное выражение в переменной `csol`.

```
C1 = symbols("C1")
csol = solveset(dsol2, C1).args[0]
csol = csol.subs(yf, y)
display(Eq(csol, C1))
```

$$-\alpha \log\left(y\right) + \beta y + \delta x - \gamma \log\left(x\right) = C_1$$

Видно, что теперь фазовая траектория для некоторого значения C_1 является соответствующей линией уровня выражения `csol`. Построив семейство линий уровня этого выражения, мы получим искомый фазовый портрет.

15 Преобразуем сначала список параметров модели `par` в словарь с соответствующими ключами. Подставим этот набор параметров в выражение `csol` и преобразуем его в функцию `Python`. Далее определяем новую сетку (существенно более подробную, чем та, которую мы использовали для построения поля направлений). Вычисляем в каждом узле сетки выражение

csol. Создаем график и строим на нем семейство линий уровня, параметр levels определяет количество линий.

```
par2 = dict(zip([al, bt, gm, dl], par))
cf = lambdify([x, y], csol.subs(par2))
xrange = np.arange(0.01, 4, 0.01)
yrange = np.arange(0.01, 4, 0.01)
X1, Y1 = np.meshgrid(xrange, yrange)
C = cf(X1, Y1)
fig, ax = plt.subplots(figsize = (6, 6))
ax.contour(X1, Y1, C, levels = 50,
           colors = "skyblue")
plt.show()
```

Результат построения показан на рис. 7.

16 Последним действием объединим фазовый портрет с полем направлений (см. рис. 8).

```
fig, ax = plt.subplots(figsize = (6, 6))
ax.contour(X1, Y1, C, levels = 50,
           colors = "skyblue")
ax.quiver(X, Y, U1, V1, color = "orangered")
plt.show()
```

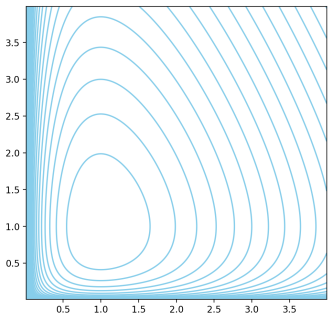


РИС. 7 Фазовый портрет системы (2)

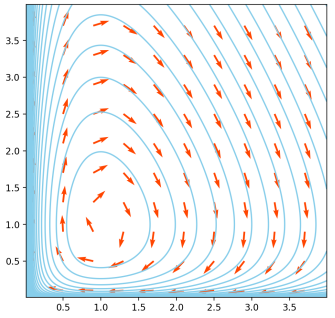


РИС. 8 Фазовый портрет системы (2), совмещенный с полем направлений

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Решите с помощью команды odeint заданную систему дифференциальных уравнений с указанными начальными условиями. Визуализируйте найденное решение в форме: интегральных кривых $x(t)$ и $y(t)$; фазовой траектории на плоскости Oxy.

1) $\dot{x} = 2y - x$, $\dot{y} = x - y$, $x_0 = 1$, $y_0 = 1$;
2) $\dot{x} = -x^2 y$, $\dot{y} = 2x - y$, $x_0 = -1$, $y_0 = 1$;
3) $\dot{x} = \sin y - x$, $\dot{y} = x^2 - y$, $x_0 = 2$, $y_0 = -2$;
4) $\dot{x} = y^2 - y^4$, $\dot{y} = 3x - 2y$, $x_0 = 1$, $y_0 = 0$.

2 Постройте с помощью команды quiver поле направлений заданной системы дифференциальных уравнений:

1) $\dot{x} = 2x + y$, $\dot{y} = x - 2y$;
2) $\dot{x} = x^2 + y^2$, $\dot{y} = x^2 - y^2$;
3) $\dot{x} = \sin x + \cos y$, $\dot{y} = \cos x + \sin y$;
4) $\dot{x} = \sin(x + y)$, $\dot{y} = y - x$.

3 Составьте и решите дифференциальное уравнение для нахождения фазовых траекторий заданной системы. Используя найденное решение, постройте с помощью команды contour фазовый портрет системы.

1) $\dot{x} = y^2 - y,\ \dot{y} = x + x^2$;
2) $\dot{x} = y + 1,\ \dot{y} = xy$;
3) $\dot{x} = \sin x \cos y,\ \dot{y} = \operatorname{tg} x \operatorname{tg} y$;
4) $\dot{x} = \dfrac{\sin y}{x},\ \dot{y} = \dfrac{\cos x}{y}$.

4 Преобразуйте заданное дифференциальное уравнение второго порядка относительно неизвестной функции $x(t)$ к системе дифференциальных уравнений первого порядка относительно двух неизвестных функций $x(t)$ и $v(t) = \dot{x}(t)$ и постройте ее фазовый портрет по аналогии с предыдущим упражнением:

1) $\ddot{x} = x - x^2$;
2) $\ddot{x}\dot{x}^2 = x$;
3) $\ddot{x}\cos\dot{x} = \sin x$;
4) $\ddot{x} = (1 - \dot{x}^3)\sin x$.

5 Постройте фазовый портрет системы (2) на всей фазовой плоскости Oxy, а не только в первой ее четверти, т. е. при $x > 0$ и $y > 0$.

6 Найдите нетривиальную точку равновесия (x_0, y_0) системы (2), лежащую в первой четверти фазовой плоскости[4]. В точке равновесия производные $\dot{x}$ и $\dot{y}$ должны быть равны нулю, это значит, что для нахождения таких точек нужно приравнять правые части `dxdt` и `dydt` системы (2) к нулю и решить полученную алгебраическую систему относительно `x` и `y`, например с помощью команды `solve`. Для рассмотренного в главе варианта модели искомая точка равновесия имеет координаты $(1, 1)$.

[4] Тривиальной точкой покоя является точка $(0, 0)$.

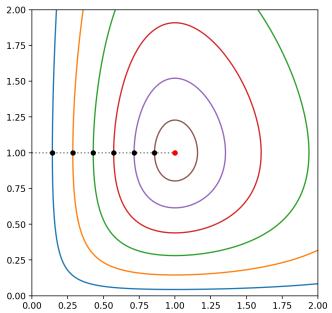


РИС. 9 Схема построения фазового портрета с помощью приближенного решения системы (2), красным маркером отмечена точка равновесия (x_0, y_0)

7 Постройте фазовый портрет системы (2) с помощью приближенного ее решения (например, командой `odeint`). Для этого на интервале $[0, x_0]$ надо расположить равномерно n точек (см. рис. 9):

$$x_i = \frac{ix_0}{n+1},\ i = 1, \ldots, n. \tag{5}$$

Для каждой выбранной точки построить приближенное решение системы с начальным условием

$$x(0) = x_i,\ y(0) = y_0 \tag{6}$$

на достаточно большом интервале времени, чтобы фазовая траектория замкнулась. Используя найденное решение, найти минимальный момент времени (период колебаний), когда траектория замыкается. Этот момент легко отслеживается сравнением координаты y решения с величиной y_0. После этого надо построить фазовую траекторию на ее периоде.

8 Используя результат выполнения предыдущего упражнения, постройте график зависимости периода колебаний системы от величины $x(0)$ при условии, что $y(0) = y_0$.

9 С помощью приближенного решения модели Лотки—Вольтерры постройте анимацию движения сразу нескольких точек вдоль соответствующих фазовых траекторий на плоскости Oxy. Используйте начальные условия из упражнения 7 (см. рис. 9). Включите в каждый кадр анимации в качестве фонового неменяющегося изображения: а) семейство фазовых траекторий, по которым движутся точки; б) поле направлений системы.

10 Более правдоподобной моделью роста одной популяции является модель Ферхюльста, учитывающая внутривидовую конкуренцию за ресурсы. Если мы модифицируем соответствующим образом второе уравнение (для жертв) в модели Лотки—Вольтерры:

$$\dot{y} = \gamma y \cdot (1 - \frac{y}{M}) - \delta xy, \tag{7}$$

то получим систему, фазовыми траекториями которой являются спирали, стремящиеся к единственной устойчивой точке равновесия (рис. 10). Постройте и визуализируйте такую модель средствами библиотек `SymPy` и `Matplotlib` по аналогии с рассмотренной в главе моделью.

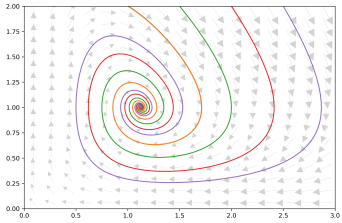


РИС. 10 Фазовый портрет в модели с внутривидовой конкуренцией в популяции жертв

11 Рассмотрим простую математическую модель межвидовой конкуренции. Пусть имеется две популяции, размеры которых равны $x(t)$ и $y(t)$. Оба вида зависят от одного и того же ресурса, доступное количество (объем) $r(t)$ которого в момент времени t задается формулой

$$r(t) = r_0 - ax - by, \tag{8}$$

где r_0 — естественное количество ресурса (при отсутствии популяций), a и b — удельные объемы потребления данного ресурса особями обоих видов. Дифференциальные уравнения, описывающие динамику роста рассматриваемых популяций, имеют одинаковый вид:

$$\begin{aligned} \dot{x} &= (\alpha_1 r - \beta_1)x, \\ \dot{y} &= (\alpha_2 r - \beta_2)y, \end{aligned} \tag{9}$$

где $\alpha_{1,2}$ — эффективности потребления ресурсов двумя видами, $\beta_{1,2}$ — коэффициенты смертности, определяющие скорости естественной убыли популяций в случае отсутствия ресурсов. То есть скорость роста популяции пропорциональна ее размеру, причем коэффициент пропорциональности (удельная скорость роста) определяется количеством доступного ресурса и коэффициентом смертности. Подставляя в (9) формулу для $r(t)$, приходим к системе

дифференциальных уравнений относительно функций $x(t)$ и $y(t)$:

$$\begin{aligned} \dot{x} &= (\alpha_1(r_0 - ax - by) - \beta_1)x, \\ \dot{y} &= (\alpha_2(r_0 - ax - by) - \beta_2)y. \end{aligned} \tag{10}$$

Преобразуем эти уравнения к более простому виду: вынесем за скобки коэффициенты $\alpha_{1,2}$ и обозначим $r_0 - \beta_{1,2}/\alpha_{1,2}$ через $\gamma_{1,2}$:

$$\begin{aligned} \dot{x} &= \alpha_1(\gamma_1 - ax - by)x, \\ \dot{y} &= \alpha_2(\gamma_2 - ax - by)y. \end{aligned} \tag{11}$$

Георгий Францевич Гаузе

Все коэффициенты в (11) являются строго положительными. Проведите качественный анализ данной модели и покажите, что независимо от начального условия ровно один из двух рассматриваемых в модели видов обязательно вымирает. Этот факт является частным случаем более общего принципа *конкурентного исключения Гаузе*: если два вида занимают одну и ту же экологическую нишу, то или один вид вымирает, или виды эволюционируют с разделением своих экологических ниш. Таким образом, если два таких вида сосуществуют, то между ними должно быть какое-то экологическое различие.

ГЛАВА 15

Системы реакций первого порядка

Задача • Кинетика отдельных химических реакций описывается дифференциальными уравнениями первого порядка. При этом уравнение материального баланса позволяет свести описание всей реакции со многими реагентами к одному дифференциальному уравнению относительно убыли одного из реагентов.

В случае когда вещества-реагенты участвуют сразу в нескольких реакциях, моделирование кинетики этих реакций приводит уже к системам дифференциальных уравнений относительно нескольких неизвестных функций, даже с учетом уравнения материального баланса. Если к тому же все реакции рассматриваемой системы являются элементарными реакциями первого порядка, т. е. имеют вид[1]

$$\mathrm{A} \to \textit{продукты},$$

то описывающая этот набор реакций система дифференциальных уравнений будет линейной и однородной. Примером такого типа системы реакций является так называемая параллельная реакция с обратимостью в одной стадии:

$$\mathrm{A} \underset{k_2}{\overset{k_1}{\rightleftharpoons}} \mathrm{B} \xrightarrow{k_3} \mathrm{C}, \qquad (1)$$

в которой вещество B превращается (параллельно) в вещества A и C, при этом первая стадия является обратимой — вещество A превращается обратно в B. Коэффициенты k_i в (1) — константы скоростей соответствующих реакций.

Система дифференциальных уравнений, описывающих набор реакций (1), строится с помощью *прин-*

[1] Продуктов реакции может быть несколько, а реагент — только один.

ципа независимости — для каждого реагента находим скорость изменения его концентрации в каждой из реакций, после чего складываем найденные скорости. Например, для вещества A скорость изменения концентрации a согласно реакции $A \to B$ равна $-k_1 a$, согласно реакции $B \to A$ эта скорость равна $k_2 b$, в третьей реакции $B \to C$ вещество A не участвует, т. е. скорость равна нулю. Складывая эти три скорости, получаем дифференциальное уравнение для $a(t)$:

$$\dot{a} = -k_1 a + k_2 b. \tag{2}$$

Выполняя описанные действия для двух других реагентов B и C с концентрациями b и c соответственно, приходим к искомой системе дифференциальных уравнений:

$$\begin{aligned} \dot{a} &= -k_1 a + k_2 b, \\ \dot{b} &= k_1 a - (k_2 + k_3) b, \\ \dot{c} &= k_3 b. \end{aligned} \tag{3}$$

Используем традиционное начальное условие:

$$a(0) = a_0,\ b(0) = b_0 \text{ и } c(0) = c_0. \tag{4}$$

В настоящей главе мы рассмотрим метод решения начальных задач для линейных систем, основанный на использовании так называемого *интегрального преобразования Лапласа*, позволяющего свести систему дифференциальных уравнений с заданными начальными условиями к решению системы алгебраических уравнений.

Преобразование Лапласа • Пусть задана некоторая функция $x(t)$, определенная при $t \geq 0$. Рассмотрим несобственный интеграл с параметром s следующего вида:

$$\int_0^\infty x(t) e^{-st} dt. \tag{5}$$

Во всех последующих выкладках будем предполагать, что все встречающиеся несобственные интегралы являются сходящимися[2]. Интеграл (5) для каждого допустимого значения s будет равен некоторому числу, т. е. он представляет собой некоторую функцию $X(s)$. Эта функция и называется преобразованием Лапласа функции x:

$$X(s) = \mathscr{L}[x(t)]. \tag{6}$$

[2] Это предположение справедливо для достаточно широкого класса функций — синусов и косинусов, многочленов, экспонент вида e^{at} при $a < s$, а также произведений этих функций. В общем случае сходимость интеграла (5) надо доказывать специально.

В формуле (6) функция X называется *образом* функции x, а функция x — *прообразом* функции X.

Для примера найдем образ функции e^{at}:

$$\mathscr{L}[e^{at}] = \int_0^\infty e^{at} \cdot e^{-st} dt = \int_0^\infty e^{(a-s)t} dt = \\ = \frac{e^{(a-s)t}}{a-s}\Big|_0^\infty = \frac{1}{s-a}, \quad (7)$$

т. к. верхняя подстановка $t = \infty$ обращает экспоненту в ноль (в силу сходимости несобственного интеграла), а нижняя $t = 0$ — в единицу. Следовательно, образом функции e^{at} является функция $1/(s-a)$. Преобразования Лапласа некоторых других функций приведены в табл. 1.

Преобразование Лапласа является линейным, т. е. образ линейной комбинации двух функций является линейной комбинацией с теми же коэффициентами образов этих функций:

$$\mathscr{L}[a_1 x_1 + a_2 x_2] = a_1 \mathscr{L}[x_1] + a_2 \mathscr{L}[x_2]. \quad (8)$$

Это свойство непосредственно вытекает из аналогичного свойства интегралов.

Вычислим теперь пребразование Лапласа *производной* функции $x(t)$, для чего воспользуемся формулой интегрирования по частям[3]:

$$\mathscr{L}[\dot{x}] = \int_0^\infty \dot{x} \cdot e^{-st} dt = \\ = x(t)e^{-st}\Big|_0^\infty + s\int_0^\infty x(t) \cdot e^{-st} dt = \\ = -x(0) + s\mathscr{L}[x]. \quad (9)$$

Аналогичным образом можно вывести формулы для производных любого порядка.

Известно, что преобразование Лапласа $g = \mathscr{L}[x]$ является взаимно однозначным, т. е. существует и обратное преобразование, переводящее функцию $X(s)$ обратно в функцию $x(t)$[4]:

$$x = \mathscr{L}^{-1}[X]. \quad (10)$$

Несложно показать, что обратное преобразование, как и прямое, также обладает свойством линейности.

ТАБЛ. 1 Преобразования Лапласа некоторых функций

$x(t)$	$X(s) = \mathscr{L}[x]$
a	$\frac{a}{s}$
t^n	$\frac{n!}{s^{n+1}}$
e^{at}	$\frac{1}{s-a}$
te^{at}	$\frac{1}{(s-a)^2}$
$t^n e^{at}$	$\frac{n!}{(s-a)^{n+1}}$
$\sin at$	$\frac{a}{s^2+a^2}$
$\cos at$	$\frac{s}{s^2+a^2}$
$x \sin at$	$\frac{2as}{(s^2+a^2)^2}$
$x \cos at$	$\frac{s^2-a^2}{(s^2+a^2)^2}$
$e^{at} \sin bt$	$\frac{b}{(s-a)^2+b^2}$
$e^{at} \cos bt$	$\frac{s-a}{(s-a)^2+b^2}$

[3] Для этого в формуле интегрирования по частям

$$\int u dv = uv - \int v du$$

полагаем $u = e^{-st}$, $dv = \dot{x}dt$, тогда $du = -se^{-st}dt$, $v = x$.

[4] Примеры обратного преобразования Лапласа можно получить из той же табл. 1.

Модель • Рассмотрим процесс решения системы (3) с помощью последовательного применения прямого и обратного преобразований Лапласа, поддержка которых включена в библиотеку SymPy.

1 Подключаем библиотеки SymPy и Matplotlib.

2 Продемонстрируем правила работы с преобразованием Лапласа в SymPy на простом примере. Создадим два положительных символа t и s. Определим выражение $x = e^{-2t}$.

```
t, s = symbols("t s", positive = True)
x = exp(-2 * t)
display(x)
```

$$e^{-2t}$$

3 Прямое преобразование Лапласа выполняется командой laplace_transform, которая принимает на вход три аргумента: преобразуемую функцию; переменную, от которой зависит эта функция; переменную, от которой должна зависеть преобразованная функция (образ). Результатом работы данной команды является кортеж, содержащий найденное преобразование и набор условий, при которых это преобразование является корректным.

```
laplace_transform(x, t, s)
```

```
(1/(s + 2), -oo, True)
```

4 Если нас интересует только образ заданной функции, то для этого можно использовать опцию noconds команды laplace_transform[5]. Сохраним образ выражения x в переменной X.

[5] Или использовать индексацию: laplace_transform(x, t, s)[0].

```
X = laplace_transform(x, t, s, noconds = True)
display(X)
```

$$\frac{1}{s+2}$$

5 Для большей компактности и ясности кода определим короткую версию команды laplace_transform и применим ее для преобразования выражения t^2.

```
def L(x):
    return laplace_transform(x, t, s,
                             noconds = True)
X = L(t ** 2)
display(X)
```

$$\frac{2}{s^3}$$

6 Обратное преобразование выполняется командой `inverse_laplace_transform`, принимающей на вход также три аргумента: функцию и две переменные. Эта команда возвращает только прообраз заданной функции без дополнительных условий. Сразу создадим короткий вариант этой команды и применим его к найденному выше выражению `X`, на выходе получаем исходное выражение t^2.

```
def invL(X):
    return inverse_laplace_transform(X, s, t)
display(invL(X))
```

$$t^2$$

7 Перейдем теперь к решению рассматриваемой модели. Определим функции $a(t)$, $b(t)$ и $c(t)$, представляющие концентрации реагентов A, B и C.

```
a, b, c = (Function(f)(t) for f in "abc")
display(a, b, c)
```

$$a(t)$$
$$b(t)$$
$$c(t)$$

8 Создаем символы для начальных концентраций.

```
a0, b0, c0 = [symbols(f + "0") for f in "abc"]
display(a0, b0, c0)
```

$$a_0$$
$$b_0$$
$$c_0$$

9 Задаем константы k_1, k_2 и k_3 скоростей химических реакций.

```
k1, k2, k3 = 2, 1, 2
print(k1, k2, k3)
```

```
2 1 2
```

10 Определяем правые части системы (3).

```
rhsa = - k1 * a + k2 * b
rhsb = k1 * a - (k2 + k3) * b
rhsc = k3 * b
display(rhsa, rhsb, rhsc)
```

$$-2a(t) + b(t)$$
$$2a(t) - 3b(t)$$
$$2b(t)$$

11 Выпишем в явном виде построенную систему дифференциальных уравнений.

```
odea = Eq(diff(a, t), rhsa)
odeb = Eq(diff(b, t), rhsb)
odec = Eq(diff(c, t), rhsc)
display(odea, odeb, odec)
```

$$\frac{d}{dt}a(t) = -2a(t) + b(t)$$
$$\frac{d}{dt}b(t) = 2a(t) - 3b(t)$$
$$\frac{d}{dt}c(t) = 2b(t)$$

12 Теперь мы должны применить к каждому уравнению построенной системы преобразование Лапласа. Команда `laplace_transform`, однако, не умеет преобразовывать производные неизвестных функций. Поэтому нам нужно самостоятельно определить команду, соответствующую формуле (9). Применим эту команду к функции $a(t)$[6].

[6] Обратите внимание, как SymPy обозначает преобразование Лапласа неопределенной функции.

```
def Ld(x, x0):
    return s * L(x) - x0
display(Ld(a, a0))
```

$$-a_0 + s\mathcal{L}_t\left[a(t)\right](s)$$

13 Применим преобразование Лапласа к каждому из трех уравнений системы. Для преобразования левых частей (производные) используем команду `Ld` с указанием соответствующих начальных условий, для преобразования правых частей — команду `L`. В результате получаем систему алгебраических уравнений относительно образов функций $a(t)$, $b(t)$ и $c(t)$.

```
eqa = Eq(Ld(a, a0), L(rhsa))
eqb = Eq(Ld(b, b0), L(rhsb))
eqc = Eq(Ld(c, c0), L(rhsc))
display(eqa, eqb, eqc)
```

$$-a_0 + s\mathcal{L}_t\left[a(t)\right](s) = -2\mathcal{L}_t\left[a(t)\right](s) + \mathcal{L}_t\left[b(t)\right](s)$$
$$-b_0 + s\mathcal{L}_t\left[b(t)\right](s) = 2\mathcal{L}_t\left[a(t)\right](s) - 3\mathcal{L}_t\left[b(t)\right](s)$$
$$-c_0 + s\mathcal{L}_t\left[c(t)\right](s) = 2\mathcal{L}_t\left[b(t)\right](s)$$

14 Для упрощения полученных формул введем специальные символы $A(s)$, $B(s)$ и $C(s)$, обозначающие образы функций $a(t)$, $b(t)$ и $c(t)$.

```
A, B, C = [Function(F)(s) for F in "ABC"]
display(A, B, C)
```

$$A(s)$$
$$B(s)$$
$$C(s)$$

15 Сформируем замену вида $\mathcal{L}_t\left[x(t)\right] \to X(s)$ и применим ее к уравнениям `eqa`, `eqb` и `eqc`.

```
Z = {L(a): A, L(b): B, L(c): C}
eqa2 = eqa.subs(Z)
eqb2 = eqb.subs(Z)
eqc2 = eqc.subs(Z)
display(eqa2, eqb2, eqc2)
```

$$-a_0 + sA(s) = -2A(s) + B(s)$$
$$-b_0 + sB(s) = 2A(s) - 3B(s)$$
$$-c_0 + sC(s) = 2B(s)$$

16 Решим с помощью команды `solve` построенную алгебраическую систему. Результатом работы данной команды является словарь, ключами которого служат имена переменных.

```
sol = solve((eqa2, eqb2, eqc2), (A, B, C))
print(sol)
```

```
A(s):(a0*s+3*a0+b0)/(s**2+5*s+4),
B(s):(2*a0+b0*s+2*b0)/(s**2+5*s+4),
C(s):(4*a0+2*b0*s+4*b0+c0*s**2+5*c0*s+4*c0)/(s**3+5*s**2+4*s)
```

17 Извлекаем из словаря отдельные решения.

```
sola, solb, solc = sol[A], sol[B], sol[C]
display(sola, solb, solc)
```

$$\frac{a_0 s + 3a_0 + b_0}{s^2 + 5s + 4}$$
$$\frac{2a_0 + b_0 s + 2b_0}{s^2 + 5s + 4}$$
$$\frac{4a_0 + 2b_0 s + 4b_0 + c_0 s^2 + 5c_0 s + 4c_0}{s^3 + 5s^2 + 4s}$$

18 Применяем обратное преобразование Лапласа к каждому из трех последних выражений и получаем решение исходной системы дифференциальных уравнений.

```
dsa = invL(sola)
dsb = invL(solb)
dsc = invL(solc)
display(dsa, dsb, dsc)
```

$$\frac{\left(a_0 - b_0 + (2a_0 + b_0)\, e^{3t}\right) e^{-4t}}{3}$$
$$\frac{\left(-2a_0 + 2b_0 + (2a_0 + b_0)\, e^{3t}\right) e^{-4t}}{3}$$
$$\frac{\left(a_0 - b_0 - 2\,(2a_0 + b_0)\, e^{3t} + 3\,(a_0 + b_0 + c_0)\, e^{4t}\right) e^{-4t}}{3}$$

19 Построим графики найденного решения для начального условия $a_0 = 1$, $b_0 = c_0 = 0$.

```
par1 = {a0: 1, b0: 0, c0: 0}
p = plot(show = False, legend = True)
for ds, lab in zip([dsa, dsb, dsc], "ABC"):
    p.extend(plot(ds.subs(par1), (t, 0, 3),
                  label = lab, show = False))
p.show()
```

Результат построения показан на рис. 1.

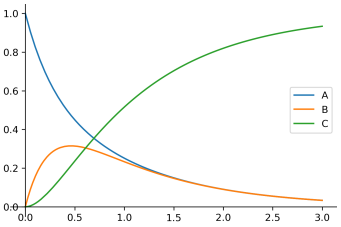


РИС. 1 Динамика системы (3) при условии, что в начальный момент времени в растворе присутствует только вещество A

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Выполните с помощью SymPy преобразование Лапласа функций, перечисленных в табл. 1.

2 Решите в SymPy с помощью преобразования Лапласа систему уравнений (3) для заданного набора параметров модели:

1) $k_1 = 3$, $k_2 = 1$, $k_3 = 4$, $a_0 = 0$, $b_0 = 1$, $c_0 = 0$;
2) $k_1 = 1$, $k_2 = 2$, $k_3 = 1$, $a_0 = 1$, $b_0 = 2$, $c_0 = 0$;
3) $k_1 = 1$, $k_2 = 20$, $k_3 = 10$, $a_0 = 1$, $b_0 = 1$, $c_0 = 0$;
4) $k_1 = 25$, $k_2 = 5$, $k_3 = 1$, $a_0 = 2$, $b_0 = 1$, $c_0 = 0$.

Динамика последней системы показана на рис. 2.

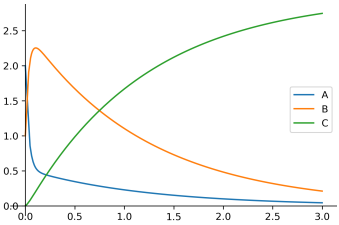


РИС. 2 Динамика системы (1) для последнего набора параметров из упражнения 2

3 В системе (3) первые два уравнения не зависят от функции $c(t)$, поэтому их можно решать отдельно, как систему из двух уравнений:

$$\begin{aligned} \dot{a} &= -k_1 a + k_2 b, \\ \dot{b} &= k_1 a - (k_2 + k_3) b. \end{aligned} \tag{11}$$

Постройте фазовый портрет такой редуцированной системы на фазовой плоскости Oab для заданного набора параметров k_1 и k_2 (см. рис. 3).

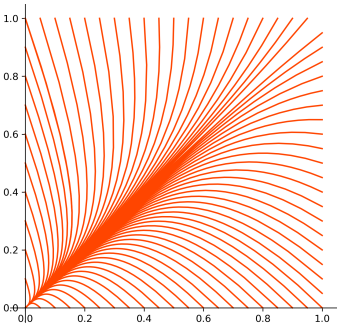


РИС. 3 Фазовый портрет системы (11) для случая $k_1 = 2$, $k_2 = 1$

4 Постройте системы дифференциальных уравнений для заданных наборов реакций первого порядка и решите их с помощью преобразования Лапласа, используя начальные условия $a_0 = 1$, $b_0 = 0$ и $c_0 = 0$ (константы скоростей для всех указанных реакций считайте равными $k = 1$):

1) $\mathrm{A} \to \mathrm{B}$;
2) $\mathrm{A} \rightleftarrows \mathrm{B}$;
3) $\mathrm{A} \to \mathrm{B} \to \mathrm{C}$;
4) $\mathrm{A} \to \mathrm{B} \to \mathrm{C} \to \mathrm{A}$.

5 Решите с помощью преобразования Лапласа следующие неоднородные дифференциальные уравнения:

1) $y' = 3y + 2$, $y(0) = 0$;
2) $y' = -y + x$, $y(0) = -1$;
3) $y' = y + e^x$, $y(0) = 1$;
4) $y' = 2y + \sin x$, $y(0) = 2$.

6 Выведите рекуррентную формулу для преобразования Лапласа k-й производной функции $x(t)$. Модифицируйте соответствующим образом введенную нами выше функцию `Ld`, добавив к ней параметр `k` и заменив начальное условие `x0` на список из k начальных условий.

7 Решите, используя результаты предыдущего упражнения, следующие начальные задачи для дифференциальных уравнений второго порядка:

1) $y'' + 3y'y + 2y = 0$, $y(0) = 1$, $y'(0) = 0$;
2) $y'' - 2y' + y = 0$, $y(0) = 0$, $y'(0) = -1$;
3) $y'' + y = 0$, $y(0) = -1$, $y'(0) = 0$;
4) $y'' - 2y' + 2y = 0$, $y(0) = 0$, $y'(0) = 1$.

8 Разработайте вариант функции `L(x)`, который был бы применим не только к выражениям, но и к уравнениям, а также корректно обрабатывал бы производные любого порядка, входящие в качестве слагаемых в заданное выражение (уравнение).

9 Обратное преобразование Лапласа рациональных выражений выполняется по следующей схеме: сначала дробь раскладывается в линейную комбинацию простых дробей[7]; после этого для каждой полученной простой дроби в таблице преобразований ищется ее прообраз; все найденные прообразы складываются с коэффициентами исходной линейной комбинации:

$$\frac{s-3}{s^2-1} = \frac{2}{s+1} - \frac{1}{s-1} \to 2e^{-t} - e^{t}. \qquad (12)$$

В `SymPy` разложение рациональной функции в сумму простых дробей выполняется с помощью команды `apart`, первым аргументом которой указывается рациональное выражение, вторым — переменная, по которой выполняется разложение:

```
out = apart((s - 3) / (s ** 2 - 1), s)
display(out)
```

$$\frac{2}{s+1} - \frac{1}{s-1}$$

10 Разложите с помощью команды `apart` следующие рациональные выражения в суммы простейших дробей.

1) $\dfrac{x}{x^3-1}$;
2) $\dfrac{8}{x^4-16}$;
3) $\dfrac{x^2+1}{x^3-x}$;
4) $\dfrac{1-2x}{x^2-4x+3}$.

[7] Этот же прием используется при вычислении интегралов от рациональных функций.

ГЛАВА 16

Геодезические линии

Задача • Классической задачей *геодезии* является нахождение кривых наименьшей длины, соединяющих две заданные точки на поверхности Земли. Такие кривые называются *геодезическими линиями*, или просто *геодезическими*.

В случае небольших расстояний[1] поверхность Земли можно считать практически плоской[2]. Тогда кратчайшей кривой между двумя точками будет, очевидно, отрезок соединяющей их прямой линии (рис. 1). Для больших расстояний нужно учитывать искривленную форму поверхности Земли. Если предположить, что этой формой является сфера, то кратчайшей кривой будет ортодрома — короткая дуга большой окружности, проходящей через две заданные точки (рис. 2).

На практике, однако, для построения геодезических кривых требуется учитывать как перепад высот в рассматриваемой местности, например наличие возвышенностей или впадин, так и тот факт, что реальной формой Земли является не сфера, а так называемый геоид. Оба этих фактора существенно усложняют процесс нахождения геодезических кривых на поверхности Земли.

В настоящей главе мы рассмотрим общий подход к построению геодезических для заданной поверхности в трехмерном пространстве, результатом применения которого является дифференциальное уравнение, описывающее семейство всех геодезических линий на данной поверхности.

Пусть поверхность задана параметрически системой соотношений

$$x = x(u, v),\ y = y(u, v),\ z = z(u, v). \qquad (1)$$

[1] За этот случай отвечает специальный раздел геодезии, называемый *топографией*.

[2] Конечно, при условии отсутствия в данном месте значительных перепадов высот, что справедливо, например, для водных поверхностей — озер, морей и т. д.

РИС. 1 Кратчайший путь на плоскости

РИС. 2 Кратчайший путь на сфере

Рассмотрим на этой поверхности две точки A и B, которым соответствуют значения параметров (u_1, v_1) и (u_2, v_2), и кривую, соединяющую указанные точки, определяемую некоторой функцией $v(u)$. Длина такой кривой равна следующему интегралу вдоль нее:

$$L = \int_A^B ds, \tag{2}$$

где ds — длина бесконечно малого элемента (дуги) данной кривой. Согласно теореме Пифагора

$$ds = \sqrt{dx^2 + dy^2 + dz^2}. \tag{3}$$

Выразим из (1) дифференциалы dx, dy и dz через дифференциалы du и dv:

$$\begin{aligned} dx &= \frac{\partial x}{\partial u}du + \frac{\partial x}{\partial v}dv, \\ dy &= \frac{\partial y}{\partial u}du + \frac{\partial y}{\partial v}dv, \\ dz &= \frac{\partial z}{\partial u}du + \frac{\partial z}{\partial v}dv, \end{aligned} \tag{4}$$

подставим эти выражения в (3):

$$ds = \sqrt{P \cdot du^2 + Q \cdot dudv + R \cdot dv^2}, \tag{5}$$

где P, Q и R — некоторые функции переменных u и v, однозначно определяемые уравнением рассматриваемой поверхности (1):

$$\begin{aligned} P(u,v) &= \left(\frac{\partial x}{\partial u}\right)^2 + \left(\frac{\partial y}{\partial u}\right)^2 + \left(\frac{\partial z}{\partial u}\right)^2, \\ Q(u,v) &= 2\left(\frac{\partial x}{\partial u}\frac{\partial x}{\partial v} + \frac{\partial y}{\partial u}\frac{\partial y}{\partial v} + \frac{\partial z}{\partial u}\frac{\partial z}{\partial v}\right), \\ R(u,v) &= \left(\frac{\partial x}{\partial v}\right)^2 + \left(\frac{\partial y}{\partial v}\right)^2 + \left(\frac{\partial z}{\partial v}\right)^2. \end{aligned} \tag{6}$$

Вынесем в (5) дифференциал du из-под корня и заменим отношение dv/du на производную v':

$$ds = \sqrt{P + Qv' + Rv'^2}du. \tag{7}$$

Таким образом, мы приходим к следующей формальной постановке нашей задачи: требуется найти

такую функцию $v(u)$, удовлетворяющую *граничным* условиям

$$v(u_1) = v_1,\ v(u_2) = v_2, \tag{8}$$

которая бы доставляла *минимум* интегралу

$$L[v] = \int_{u_1}^{u_2} \sqrt{P + Qv' + Rv'^2}\,du \tag{9}$$

для заданных функций $P(u, v)$, $Q(u, v)$ и $R(u, v)$.

Уравнение Эйлера—Лагранжа • Интеграл вида (9) называется *функционалом*, неформально говоря — функцией от функции. Задача поиска функции, доставляющей экстремум заданному функционалу, является стандартной задачей *вариационного исчисления*. Рассмотрим такую задачу для функционала следующего вида:

$$L[y] = \int_{x_1}^{x_2} F(x, y, y')dx, \tag{10}$$

значение которого зависит от заданной функции $y(x)$ и ее производной $y'(x)$[3]. Дополнительно потребуем, чтобы на концах интервала интегрирования функция $y(x)$ принимала заданные значения y_1 и y_2 соответственно:

$$y(x_1) = y_1,\ y(x_2) = y_2. \tag{11}$$

[3] Всюду ниже мы будем предполагать нужную степень гладкости рассматриваемых функций, как минимум их дифференцируемость.

Из теории вариационного исчисления известно, что функция является экстремалью функционала (10), если она удовлетворяет уравнению Эйлера—Лагранжа:

$$\frac{\partial F}{\partial y} - \frac{d}{dx}\frac{\partial F}{\partial y'} = 0. \tag{12}$$

Уравнение (12) является дифференциальным уравнением второго порядка, вместе с дополнительными *граничными* условиями (11) оно составляет так называемую *краевую задачу*.

Важным частным случаем является задача минимизации функционала (10), подынтегральная функция F в котором не зависит явным образом от переменной x:

$$L[y] = \int_{x_1}^{x_2} F(y, y')dx. \tag{13}$$

Можно показать, что для функционалов такого вида уравнение Эйлера—Лагранжа сводится к более простому дифференциальному уравнению первого порядка, называемому тождеством Бельтрами:

$$F - y' \frac{\partial F}{\partial y'} = C_1, \tag{14}$$

где C_1 — некоторая произвольная константа.

Эудженио Бельтрами

Модель • Рассмотрим задачу построения геодезических кривых на поверхности прямого кругового конуса с помощью библиотеки `SymPy`. Поверхность такого конуса в цилиндрических координатах описывается уравнением

$$z = \gamma r, \tag{15}$$

где коэффициент $\gamma > 0$ определяет угол раствора конуса. Следовательно, мы можем параметризовать коническую поверхность с помощью двух полярных координат r и φ:

$$x = r\cos\varphi,\ \ y = r\sin\varphi,\ \ z = \gamma r. \tag{16}$$

Для краткости изложения рассмотрим фиксированное значение параметра $\gamma = 4$.

1 Подключаем библиотеки `SymPy`, `NumPy` и `Matplotlib`.

2 Создаем символы r и φ, задаем значение параметра γ и определяем формулы (16).

```
r, phi = symbols("r phi")
gamma = 4
x = r * cos(phi)
y = r * sin(phi)
z = gamma * r
display(x, y, z)
```

$$r\cos(\varphi)$$
$$r\sin(\varphi)$$
$$4r$$

3 Вычисляем частные производные выражений (16) по φ и r, с помощью которых согласно формулам (6) определяем функции P, Q и R. В качестве переменной u в нашем случае выступает параметр φ, в качестве переменной v — параметр r.

```
dxdr, dxdp = diff(x, r), diff(x, phi)
dydr, dydp = diff(y, r), diff(y, phi)
dzdr, dzdp = diff(z, r), diff(z, phi)
P = dxdp ** 2 + dydp **2 + dzdp ** 2
Q = dxdr * dxdp + dydr * dydp + dzdr * dzdp
R = dxdr ** 2 + dydr **2 + dzdr ** 2
display(P, Q, R)
```

$$r^2 \sin^2(\varphi) + r^2 \cos^2(\varphi)$$

$$0$$

$$\sin^2(\varphi) + \cos^2(\varphi) + 16$$

4 Введем символ `rf` для обозначения положительной функции $r(\varphi)$ и символ `rf_` для производной этой функции. Определим подынтегральное выражение `F` согласно (9), заменив в этом выражении символ переменной `r` на символ функции `rf`, и упростим полученное выражение[4].

[4] Нетрудно показать, что «волшебная» константа 17 в полученном выражении равна $1 + \gamma^2$.

```
rf = Function("r", positive = True)(phi)
rf_ = diff(rf, phi)
F = sqrt(P + Q * rf_ + R * rf_ ** 2)
F = F.subs(r, rf).simplify()
display(F)
```

$$\sqrt{r^2(\varphi) + 17\left(\frac{d}{d\varphi} r(\varphi)\right)^2}$$

5 Введем два параметра (две константы интегрирования) k и α, первый из которых будем использовать вместо параметра C_1 в тождестве Бельтрами (14), а второй — при решении соответствующего дифференциального уравнения. Создаем, упрощаем и сохраняем в переменной `B` уравнение Бельтрами.

```
k, alpha = symbols("k alpha", positive = True)
B = Eq(F - rf_ * diff(F, rf_), k).simplify()
display(B)
```

$$k = \frac{r^2(\varphi)}{\sqrt{r^2(\varphi) + 17\left(\frac{d}{d\varphi} r(\varphi)\right)^2}}$$

6 Разрешаем полученное соотношение относительно производной `rf_`.

```
sol = solve(B, rf_)
display(sol)
```

```
-sqrt(17)*sqrt((-k**2+r(phi)**2)*r(phi)**2)/(17*k)
sqrt(17)*sqrt(-(k**2-r(phi)**2)*r(phi)**2)/(17*k)
```

7 Из двух найденных решений выбираем решение со знаком плюс и составляем искомое дифференциальное уравнение.

```
ode = Eq(rf_, sol[1])
display(ode)
```

$$\frac{d}{d\varphi}r(\varphi) = \frac{\sqrt{-17k^2 + 17r^2(\varphi)}r(\varphi)}{17k}$$

8 Находим общее решение построенного дифференциального уравнения.

```
dsol = dsolve(ode, rf)
display(dsol)
```

$$r(\varphi) = \begin{cases} \frac{k}{\cos\left(C_1 k - \frac{\sqrt{17}\varphi}{17}\right)} & \text{for } k > \frac{k}{\cos\left(C_1 k - \frac{\sqrt{17}\varphi}{17}\right)} \\ \text{NaN} & \text{otherwise} \end{cases}$$

9 Используя несколько раз атрибут `args`, извлекаем из выражения `dsol` функцию, представляющую решение дифференциального уравнения.

```
rsol = dsol.args[1].args[0].args[0]
display(rsol)
```

$$\frac{k}{\cos\left(C_1 k - \frac{\sqrt{17}\varphi}{17}\right)}$$

10 Заменим в найденном решении константу C_1 на отношение α/k (оба параметра являются произвольными константами, т. е. и их отношение также будет произвольной константой) и выпишем окончательное решение в терминах символов r и φ.

```
C1 = symbols("C1")
rsol = rsol.subs(C1, alpha / k)
display(Eq(r, rsol))
```

$$r = \frac{k}{\cos\left(\alpha - \frac{\sqrt{17}\varphi}{17}\right)}$$

11 Построим график полученной зависимости $r(\varphi)$ при фиксированных значениях параметров $k = 1$ и $\alpha = 0$ (см. рис. 3).

```
par = {k: 1, alpha: 0}
rs = rsol.subs(par)
plot(rs, (phi, -8 * pi, 8 * pi), ylim = (-10, 10))
```

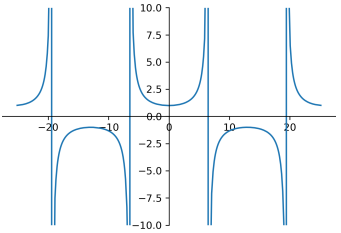


РИС. 3 График зависимости $r(\varphi)$, вертикальные линии соответствуют асимптотам

Заметим, что параметр r по своему определению (полярный радиус) является положительной величиной, поэтому геометрический смысл имеют только части графика 3, расположенные в его верхней полуплоскости.

12 Построим кривую $r(\varphi)$ в полярных координатах на плоскости Oxy в диапазоне от $-\varphi_0$ до φ_0, где φ_0 задает положение первой положительной асимптоты графика на рис. 3. Несложно убедиться, что период T функции $r(\varphi)$ равен периоду ее знаменателя, т. е.

$$T = 2\pi\sqrt{1+\gamma^2}. \tag{17}$$

Тогда

$$\varphi_0 = \frac{T}{4} = \frac{\pi\sqrt{1+\gamma^2}}{2}. \tag{18}$$

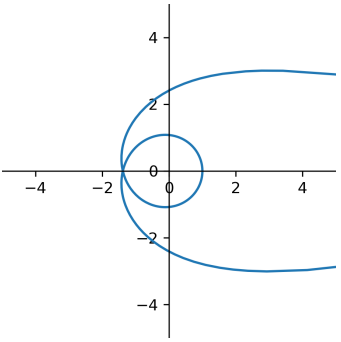


РИС. 4 График зависимости $r(\varphi)$ на плоскости Oxy для $\varphi \in (-\varphi_0, \varphi_0)$

```
phi0 = pi * sqrt(1 + gamma ** 2) / 2
xs, ys = x.subs(r, rs), y.subs(r, rs)
plot_parametric((xs, ys), (phi, -phi0, phi0),
                xlim = (-5, 5), ylim = (-5, 5),
                aspect_ratio = (1, 1))
```

13 Так как отношение периода функции $r(\varphi)$ к «естественному» периоду 2π полярной системы координат является иррациональным, то никакие из ветвей графика на рис. 3 при их визуализации на плоскости Oxy не должны совпадать друг с другом. Построим на одном графике несколько таких кривых (рис. 5). Чтобы избежать прорисовки прямых линий, соединяющих разные ветви, изменим пределы `a` и `b` каждого периода на небольшую величину ± 0.01.

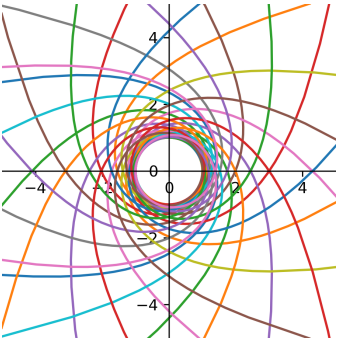


РИС. 5 Кривые, описываемые выражением `rsol` при $k = 1$, $\alpha = 0$

```
p1 = plot(xlim = (-5, 5), ylim = (-5, 5),
          aspect_ratio = (1, 1), show = False)
for i in range(-17, 17, 2):
```

```
    a = i * phi0 + 0.01
    b = (i + 2) * phi0 - 0.01
    xs, ys = x.subs(r, rs), y.subs(r, rs)
    p = plot_parametric((xs, ys), (phi, a, b),
                        show = False)
    p1.extend(p)
p1.show()
```

14 Перейдем теперь к визуализации найденных геодезических кривых на поверхности конуса с помощью библиотеки `Matplotlib`. Для этого сначала преобразуем выражения `x`, `y` и `z` в функции `Python`, подставив в них вместо символа `r` частное решение `rs` дифференциального уравнения.

```
X = lambdify([phi], x.subs(r, rs))
Y = lambdify([phi], y.subs(r, rs))
Z = lambdify([phi], z.subs(r, rs))
print(X(0), Y(0), Z(0))
```

```
1.0 0.0 4.0
```

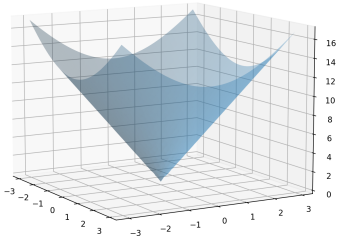


РИС. 6 Поверхность конуса

15 Построим график поверхности рассматриваемого конуса (см. рис. 6). Для этого перепишем уравнение поверхности конуса $z = \gamma r$ в декартовых координатах:

$$z = \gamma\sqrt{x^2 + y^2}. \tag{19}$$

В первой строке подключаем библиотеку для работы с трехмерными графиками. Далее создаем сетку `SX`, `SY` и на этой сетке вычисляем z-координаты точек согласно (19). После этого создаем трехмерный график и строим поверхность, заданную массивами `SX`, `SY` и `SZ`. Параметр `alpha` определяет прозрачность поверхности. Команда `view_init` в предпоследней строке задает трехмерную ориентацию графика[5].

[5] Аргументами данной команды служат два угла поворота в градусах.

```
from mpl_toolkits.mplot3d import Axes3D
xrange = np.arange(-3.0, 3.0, 0.05)
SX, SY = np.meshgrid(xrange, xrange)
SZ = gamma * np.sqrt(SX ** 2 + SY ** 2)
fig = plt.figure(figsize = (8, 8))
ax = fig.add_subplot(111, projection = "3d")
ax.plot_surface(SX, SY, SZ, alpha = 0.3)
ax.view_init(10, -30)
plt.show()
```

16 Теперь построим трехмерную кривую, заданную параметрически функциями `X`, `Y` и `Z`. Чтобы рисуемая

кривая не вышла за пределы графика, найдем минимальный угол φ_1, при котором функция $z(\varphi)$ принимает значение, равное максимальному элементу в массиве SZ (т. е. верхнему пределу по оси z нашего графика). Далее определяем диапазон Phi значений угла φ (от $-\varphi_1$ до φ_1), создаем трехмерный график и строим искомую кривую с помощью стандартной команды plot.

```
g = float(sqrt(1 + gamma ** 2))
phi1 = g * np.arccos(gamma / np.max(SZ))
Phi = np.linspace(-phi1, phi1, 200)
fig = plt.figure(figsize = (8, 8))
ax = fig.add_subplot(111, projection = "3d")
ax.plot(X(Phi), Y(Phi), Z(Phi),
        color = "red", lw = 2)
ax.view_init(10, -30)
plt.show()
```

Результат построения приведен на рис. 7.

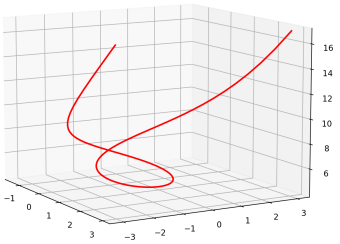


РИС. 7 Геодезическая кривая

17 Последним шагом объединим на одном рисунке построенные выше графики поверхности конуса и геодезической кривой (см. рис. 8).

```
fig = plt.figure(figsize = (8, 8))
ax = fig.add_subplot(111, projection = "3d")
ax.plot_surface(SX, SY, SZ, alpha = 0.25)
ax.plot(X(Phi), Y(Phi), Z(Phi),
        color = "red", lw = 2)
ax.view_init(10, -30)
plt.show()
```

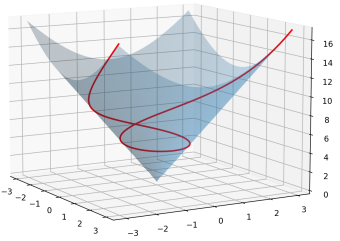


РИС. 8 Геодезическая кривая на поверхности конуса

УПРАЖНЕНИЯ И ЗАМЕЧАНИЯ

1 Найденное нами в пункте 10 уравнение геодезической кривой зависит от двух произвольных параметров (констант интегрирования) k и α. Очевидно, что параметр k является масштабным множителем, а параметр α задает поворот всей кривой относительно оси Oz. Это значит, что для заданной поверхности (при фиксированном значении γ) все геодезические линии являются подобными, они отличаются друг от друга только размером и углом поворота. Конкретные значения данных двух параметров могут быть найдены из дополнительных граничных условий, что искомая геодезическая кривая проходит через две заданные точки:

$$r(\varphi_1) = r_1,\ r(\varphi_2) = r_2. \tag{20}$$

Для рассматриваемой конической поверхности, однако, соответствующая алгебраическая система в общем случае не решается аналитически, а может быть решена только приближенными методами.

2 Исследуйте, как влияет на внешний вид геодезических кривых изменение параметра γ.

3 При построении трехмерных графиков с помощью библиотеки `Matplotlib` в системе `Jupyter Notebook` может оказаться более удобным интерактивный режим их показа, когда пользователь может прямо при просмотре графика менять углы его поворота. Для этого надо первой строкой кода указать так называемую «магическую» команду

```
%matplotlib qt
```

и перезапустить ядро блокнота. После этого все графики будут строиться в отдельных окнах (рис. (9)), а для трехмерных графиков появится возможность их вращения.

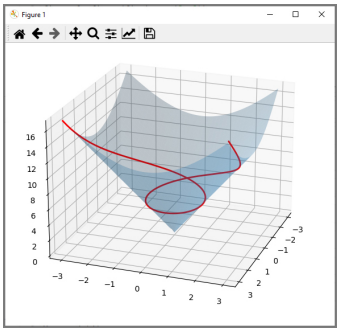


РИС. 9 Окно интерактивного показа трехмерных графиков в `Jupyter Notebook`

4 Для заданной функции $F(x, y')$ постройте и решите краевую задачу для уравнения Эйлера—Лагранжа с учетом указанных граничных условий. Так как подынтегральная функция не зависит явным образом от y, то уравнение Эйлера—Лагранжа сразу может быть сведено к дифференциальному уравнению первого порядка:

$$-\frac{d}{dx}\frac{\partial F}{\partial y'} = 0 \Rightarrow \frac{\partial F}{\partial y'} = C_1. \tag{21}$$

1) $F(x, y') = x^2 y'^2$, $y(1) = 1$, $y(2) = 2$;
2) $F(x, y') = y' + xy'^2$, $y(1) = 0$, $y(e) = 1$;
3) $F(x, y') = \sqrt{x^2 + y'^2}$, $y(0) = 1$, $y(1) = 2$;
4) $F(x, y') = y'^2 + y' \sin x$, $y(0) = 0$, $y(\pi/2) = 1$.

5 Найдите общее решения уравнение Эйлера—Лагранжа, записанного в форме тождества Бельтрами, для заданной функции $F(y, y')$:

1) $F(y, y') = yy'^2$;
2) $F(y, y') = e^y y'^3$;
3) $F(y, y') = \dfrac{y'^2}{\sqrt{y}}$;
4) $F(y, y') = \dfrac{2y + 1}{y'}$.

6 Найдите общее решения уравнение Эйлера—Лагранжа для заданного функционала $L[y]$:

1) $L[y] = \int_a^b (y^2 + y\sin x - y'^2)dx;$

2) $L[y] = \int_a^b (xe^x y' + y^2 + 4y'^2)dx;$

3) $L[y] = \int_a^b (y\ln x + y'^2)dx;$

4) $L[y] = \int_a^b (y'^3 - xy' + y)dx.$

7 Докажите с помощью решения уравнения Эйлера—Лагранжа, что геодезическими кривыми на плоскости являются прямые линии.

8 Постройте уравнение семейства геодезических кривых на поверхности прямого кругового цилиндра радиуса 1, уравнение которой в цилиндрических координатах тривиально: $r = 1$ (рис. 10). Поэтому для параметризации данной поверхности можно использовать полярный угол φ и высоту z:

$$x = \cos\varphi,\ y = \sin\varphi,\ z = z. \tag{22}$$

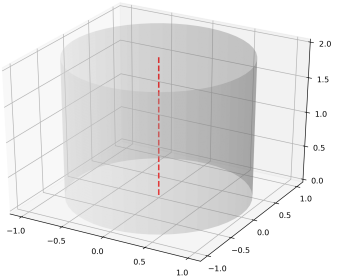


РИС. 10 Цилиндрическая поверхность

9 Выведите уравнение геодезических кривых на сферической поверхности единичного радиуса с центром в начале координат. Для построения функционала используйте уравнение сферы в сферических координатах (рис. 11)

$$r = 1$$

и соотношения между сферическими и декартовыми координатами:

$$\begin{aligned} x &= r\sin\theta\cos\varphi, \\ y &= r\sin\theta\sin\varphi, \\ z &= r\cos\theta. \end{aligned}$$

Покажите, что полученное уравнение геодезической кривой в самом деле соответствует большим окружностям сферы.

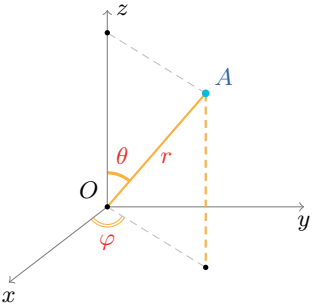


РИС. 11 Сферические координаты (r, φ, θ) точки A

10 Для поверхности, заданной уравнением $z = f(x, y)$, постройте семейство ее геодезических линий, исходящих из заданной точки (x_0, y_0). Параметризация (1) такой поверхности тривиальна:

$$x = x,\ y = y,\ z = f(x, y). \tag{23}$$

Преобразуйте соответствующее дифференциальное уравнение Эйлера—Лагранжа (второго порядка) к системе двух

дифференциальных уравнений первого порядка и решите эту систему с помощью какого-нибудь приближенного метода. Заметим, что искомое семейство геодезических кривых является однопараметрическим, его параметром k служит начальное значение производной: $y'(x_0) = k$.

1) $z = y^2$;
2) $z = x^2 + y^2$;
3) $z = \sin x$;
4) $z = y + x^3$.

11 Постройте с помощью `Matplotlib` графики следующих поверхностей:

1) $z = (x^2 + y^2 - 3)^2$;
2) $z = x^2 \sin y$;
3) $z = \dfrac{1}{1 + x^2 + y^2}$;
4) $z = \cos(x + y) + 2\cos(x - y)$.

Книги издательства «ДМК ПРЕСС»
можно купить оптом и в розницу
в книготорговой компании «Галактика»
(представляет интересы издательств
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).
Адрес: г. Москва, пр. Андропова, 38;
тел.: **(499) 782-38-89**, электронная почта: **books@alians-kniga.ru**.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.

Ершов Николай Михайлович

Практическое введение
в решение дифференциальных уравнений в Python

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Зам. главного редактора *Сенченкова Е. А.*
Корректор *Синяева Г. И.*
Верстка *Ершов Н. М.*
Дизайн обложки *Мовчан А. Г.*

Гарнитура Computer Modern LaTeX. Печать цифровая.
Усл. печ. л. 14,3. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**

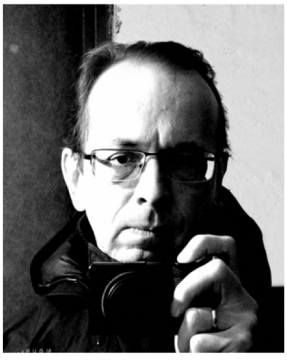
О чем эта книга

Настоящая книга посвящена вопросам практического применения символьных вычислений для решения дифференциальных уравнений и их систем с использованием библиотеки символьных вычислений SymPy языка программирования Python. В каждой главе рассматривается какая-нибудь одна прикладная модель из области физики, химии, биологии и т. д. После теоретического вывода возникающих в модели дифференциальных уравнений максимально детально описывается процесс формализации модели и решения соответствующих дифференциальных уравнений с использованием SymPy. Особое внимание при этом уделяется анализу и визуализации найденных решений с помощью библиотеки Matplotlib. Изложение материала сопровождается большим числом иллюстраций и упражнений.

Издание ориентировано на широкий круг читателей, интересующихся проблемами математического моделирования.

С помощью этой книги

- научитесь выполнять формализацию прикладных моделей и строить дифференциальные уравнения, описывающие их поведение;
- освоите аналитические и приближенные методы решения дифференциальных уравнений и их систем с использованием библиотеки символьных вычислений SymPy;
- научитесь визуализировать полученные решения дифференциальных уравнений с помощью библиотеки Matplotlib.



Николай Ершов

Старший научный сотрудник факультета вычислительной математики и кибернетики МГУ им. М. В. Ломоносова, а также доцент института системного анализа и управления университета «Дубна». Область научных интересов: математическое и компьютерное моделирование, естественные вычисления, дискретная оптимизация, распознавание образов.

ISBN 978-5-93700-147-4

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК «Галактика»
books@alians-kniga.ru

издательство
www.дмк.рф