8. ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ И СИСТЕМ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

Обыкновенным дифференциальным уравнением (ОДУ) порядка n называется уравнение вида $F(x, y(x), y'(x), y''(x),..., y^{(n)}(x)) = 0$, где n – порядок наивысшей производной, входящей в уравнение. В каноническом виде ОДУ n-го порядка записывается как $y^{(n)}(x) = f(x, y(x), y'(x), y''(x),..., y^{(n-1)}(x))$. Под интегрированием дифуравнения понимают нахождение функции $y(x)$, которая удовлетворяет этому уравнению. $y(x)$ называется решением дифференциального уравнения. Общее решение ОДУ n-го порядка имеет вид $y = y(x, c_1, c_2,..., c_n)$, где c_1, c_2, … , c_n – произвольные константы.

Системой обыкновенных дифференциальных уравнений первого порядка называется система уравнений вида

$$\begin{cases} \dfrac{dy_1(x)}{dx} = f_1(x, y_1(x), y_2(x),..., y_n(x)) \\ \dfrac{dy_2(x)}{dx} = f_2(x, y_1(x), y_2(x),..., y_n(x)) \\ ..... \\ \dfrac{dy_n(x)}{dx} = f_n(x, y_1(x), y_2(x),..., y_n(x)) \end{cases}.$$

Здесь x – независимая переменная; f_1, f_2, …, f_n – заданные функции. Решением этой системы называют совокупность функций $y_1(x) = \varphi_1(x),..., y_n(x) = \varphi_n(x)$, которые после подстановки в систему уравнений обращают их в тождества.

Дифференциальные уравнения и системы дифференциальных уравнений имеют бесчисленное множество решений, отличающихся друг от друга набором констант. Для того чтобы выделить среди бесчисленного множества решений одно решение, необходимо задать начальные и/или граничные условия. Количество таких условий должно совпадать с порядком дифференциального уравнения или с количеством дифференциальных уравнений в системе дифуравнений. В зависимости от вида этих дополнительных условий различают *задачу Коши* и *краевую задачу*.

Задача Коши для ОДУ первого порядка ставится так: найти решение дифференциального уравнения $y' = f(x, y)$, проходящее через точку с координатами (x_0,y_0), где x_0 и y_0 – некоторые заданные числа:

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases}.$$

Задача Коши для дифференциального уравнения порядка n есть задача о нахождении частного решения уравнения, удовлетворяющего начальным условиям $y(x_0) = y_0, y'(x_0) = y'_0, ..., y^{(n-1)}(x_0) = y_0^{(n-1)}$. Здесь $y_0, y'_0, ..., y_0^{(n-1)}$ – некоторые заданные числа.

Графическое изображение частного решения называют интегральной кривой. Общее решение дифференциального уравнения n-го порядка определяет n-параметрическое семейство интегральных кривых.

8.1. Решение задачи Коши для дифференциального уравнения первого порядка

Если дифференциальное уравнение не удается решить точными методами, его решают численно. Рассмотрим численные методы решения задачи Коши для ОДУ первого порядка:

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases}. \qquad (8.1)$$

Результатом численного решения дифференциального уравнения является таблица:

x	x_0	x_1	x_2	…	x_n
y	y_0	y_1	y_2	…	y_n

Первая точка таблицы задана в условии задачи. Задавая шаг h_i, вычисляют значения x_1, x_2, … , x_n: $x_{i+1}=x_i+h_i$, i=0, 1, … , n-1. Шаг может быть как переменным, так и постоянным. y_i вычисляют по формулам, соответствующим методу решения. Рассмотрим некоторые из них.

Метод Эйлера. В методе Эйлера значения $y_{i+1} = y(x_{i+1})$ вычисляют по формуле: $y_{i+1} = y_i + h_i f(x_i, y_i), i = \overline{0, n-1}$. Метод Эйлера довольно простой, но не очень точный. Его погрешность порядка $o(h)$.

Метод Эйлера называют методом ломаных. В начальной точке (x_0, y_0) проведем прямую с угловым коэффициентом k_0. $k_0 = y'(x_0) = f(x_0, y_0) = tg\alpha_0$ в силу дифференциального уравнения (8.1). Эта прямая является касательной к (неизвестной) интегральной кривой – графику искомого решения $y = \varphi(x)$ задачи Коши. Уравнение касательной $y - y_0 = k_0(x - x_0)$ или $y = y_0 + f(x_0, y_0)(x - x_0)$. В качестве приближенного решения задачи Коши (8.1) в узле x_1 примем ординату y_1 точки пересечения касательной с вертикальной прямой $x = x_1$:

$$y_1 = y_0 + f(x_0, y_0)(x_1 - x_0) = y_0 + h_0 f(x_0, y_0).$$

Через точку (x_1, y_1) проведем прямую с угловым коэффициентом $k_1 = y'(x_1) = f(x_1, y_1) = tg\alpha_1$. Ее уравнение $y = y_1 + f(x_1, y_1)(x - x_1)$. Эта прямая уже не является касательной к интегральной кривой $y = \varphi(x)$, т. к. точка (x_1, y_1) не лежит на ней. Пересечение этой прямой с вертикалью $x = x_2$ имеет ординату $y_2 = y_1 + f(x_1, y_1) \cdot (x_2 - x_1) = y_1 + h_1 f(x_1, y_1)$.

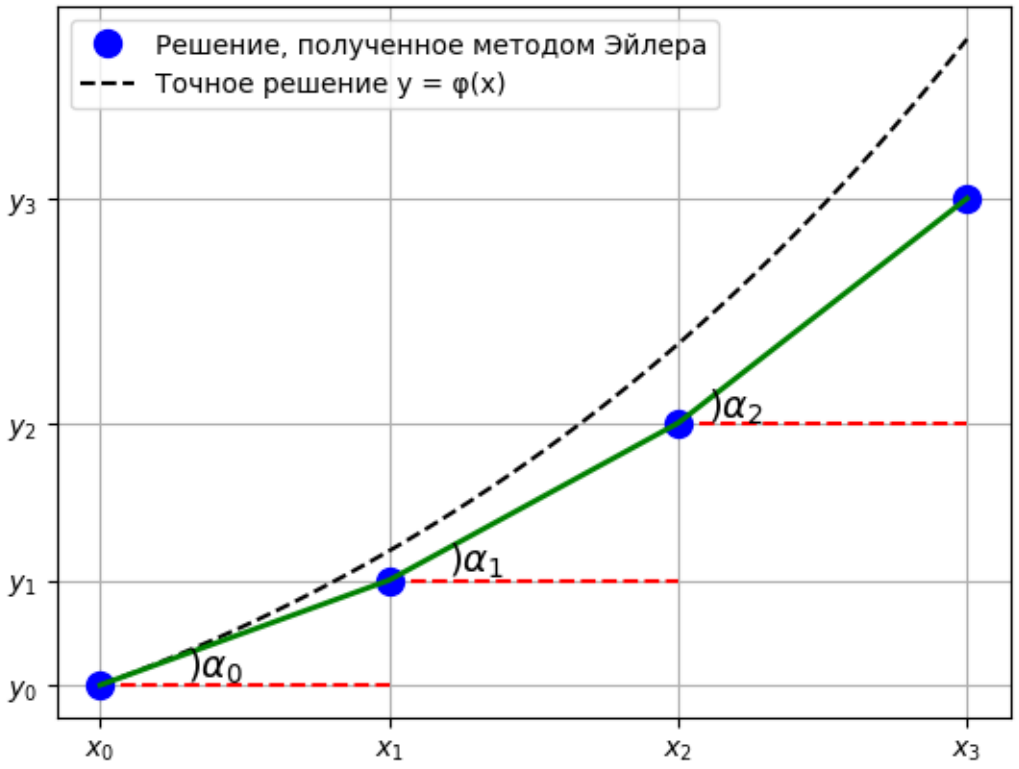


Рис. 8.1. Графическая иллюстрация метода Эйлера

Через точку (x_2, y_2) проведем прямую с угловым коэффициентом $k_2 = y'(x_2) = f(x_2, y_2) = tg\alpha_2$. Ее уравнение $y = y_2 + f(x_2, y_2)(x - x_2)$. Пересечение этой прямой с вертикалью $x = x_3$ имеет ординату

$$y_3 = y_2 + f(x_2, y_2) \cdot (x_3 - x_2) = y_2 + h_2 f(x_2, y_2).$$

Работу метода рассмотрим на примере.

Пример 8.1. Решить задачу Коши для ОДУ первого порядка $\begin{cases} y' = \sin x + y \\ y(0) = 2 \end{cases}$ методом Эйлера. Сделать три шага (h=0.1). Сравнить полученное решение с известным точным решением задачи $y = \frac{5}{2}e^x - \frac{\sin x}{2} - \frac{\cos x}{2}$. Результаты свести в таблицу.

Решение. В соответствии с формулой метода Эйлера $y(x_1) = y(x_0) + hf(x_0, y_0)$. Здесь x_0=0, y_0=2, h=0.1, $f(x,y)$=sin(x)+y. Тогда $y_1 = y(x_1) = 2 + 0.1(\sin 0 + 2) = 2.2$. Находим следующую точку. x_1=x_0+h=0.1, $y(x_2) = y(x_1) + hf(x_1, y_1)$. $y_2 = y(x_2) = 2.2 + 0.1(\sin 0.1 + 2.2) = 2.429983$. Третий шаг: x_2=x_1+h=0.2, $y_3 = y(x_3) = 2.429983 + 0.1(\sin 0.2 + 2.429983) = 2.692848$. Вычисляем точное значение функции в точках 0.1, 0.2 и 0.3 и результаты сводим в табл. 8.1:

Таблица 8.1

Результаты решения ОДУ методом Эйлера

x	y (метод Эйлера)	y точное
0	2	2
0.1	2.2	2.2155085
0.2	2.429983	2.46413894
0.3	2.692848	2.74921867

Из таблицы видно, что по мере удаления от начальной точки абсолютная погрешность решения возрастает.

Метод Эйлера — представитель *одношаговых* приближенных методов, в которых решение в (i +1)-м узле получается на основе решения только в *одном* предыдущем i-м узле. Тем самым информация о более ранних уже вычисленных значениях игнорируется. «Расточительный» подход к получаемым результатам оборачивается повышенным объемом вычислений. Одношаговые методы не самые экономичные в этом смысле.

Как и в любом одношаговом методе, начиная со второго шага исходное значение y_i в формуле $y(x_{i+1}) = y(x_i) + h_i f(x_i, y_i)$ само́ является приближенным, т. е. погрешность на каждом последующем шаге систематически возрастает.

Уменьшение h повышает точность вычислений, но резко увеличивает количество вычислений. В целом метод ломаных Эйлера применим только для грубой прикидки.

Рассмотрим более точный метод решения задачи Коши для ОДУ первого порядка.

Метод Рунге-Кутта. Это метод, в котором следующее значение функции y_{i+1} вычисляется по формуле: $y_{i+1} = y_i + \frac{k_{1i} + 2k_{2i} + 2k_{3i} + k_{4i}}{6}$, где

$$k_{1i} = h_i f(x_i, y_i),$$

$$k_{2i} = h_i f(x_i + \frac{h_i}{2}, y_i + \frac{k_{1i}}{2}),$$

$$k_{3i} = h_i f(x_i + \frac{h_i}{2}, y_i + \frac{k_{2i}}{2}),$$

$$k_{4i} = h_i f(x_i + h, y_i + k_{3i}).$$

Решим пример 8.1 методом Рунге-Кутта.

$$y_1 = y_0 + \frac{k_{10} + 2k_{20} + 2k_{30} + k_{40}}{6},\ k_{10} = hf(x_0, y_0) = 0.1(\sin(0) + 2) = 0.2;$$

$$k_{20} = hf(x_0 + \frac{h}{2}, y_0 + \frac{k_{10}}{2}) = 0.1 f\left(0.05; 2 + \frac{0.2}{2}\right) = 0.1 f(0.05, 2.1) = 0.1(\sin 0.05 + 2.1) =$$

$$= 0.2149979;\ k_{30} = hf(x_0 + \frac{h}{2}, y_0 + \frac{k_{20}}{2}) = 0.1 f\left(0.05; 2 + \frac{0.2149979}{2}\right) =$$

$$= 0.1 f(0.05, 2.107499) = 0.1(\sin 0.05 + 2.107499) = 0.2157478;\ k_{40} = hf(x_0 + h, y_0 + k_{30}) =$$

$$= 0.1 f(0.1, 2 + 0.2157478) = 0.1(\sin(0.1) + 2.2157478) = 0.2315581.$$

$$y_1 = 2 + \frac{0.22 + 2 \cdot 0.2149979 + 2 \cdot 0.2157478 + 0.2315581}{6} = 2.2155083.$$

Аналогично можно найти y_2 и y_3: y_2=2.46413841, y_3= 2.74921778.

Таблица 8.2.

Результаты решения ОДУ методом Рунге-Кутта

x	y (метод Рунге-Кутта)	y точное
0	2	2
0.1	2.2155083	2.2155085
0.2	2.46413841	2.46413894
0.3	2.74921778	2.74921867

Вывод: метод Рунге-Кутта значительно точнее метода Эйлера. Порядок точности метода Рунге-Кутта $o(h^4)$. За точность приходится платить значительно бОльшим количеством вычислений.

Подведем некоторые итоги.

Метод Эйлера более простой, чем метод Рунге – Кутта, но уступает ему в точности. Оба метода решения ДУ – метод Эйлера, метод Рунге – Кутта – являются одношаговыми. Напомним, что это означает построение y_{i+1} на основе только y_i с игнорированием более ранних предшествующих результатов. Все одношаговые методы сопряжены с избыточными вычислениями, объем которых можно существенно уменьшить при использовании полученных ранее результатов.

Альтернативой одношаговым являются многошаговые методы интегрирования ДУ – семейство методов Адамса. В этих методах для вычисления y_{i+1} используется несколько значений приближенного решения на предыдущих шагах: y_i , y_{i-1}, y_{i-2}, … . Поскольку к моменту вычисления y_{i+1} они уже найдены, можно избежать многочисленных вычислений значений $f(x, y)$. Но, чтобы проводить вычисления по методу Адамса, первые значения y_0, y_1, … приходится находить одношаговыми методами. В целом методы Адамса в несколько раз менее трудоемки, чем метод Рунге – Кутта.

Оба метода нетрудно запрограммировать. Ниже приведен код решения рассмотренной задачи обоими методами. Сделан 21 шаг:

```
import numpy as np; import matplotlib.pyplot as plt
def f(x,y):
  z=np.sin(x)+y
  return z
p=np.array([0,2]);a=0;b=3;n=21
s=p[1];x=np.linspace(a,b,n);h=(b-a)/(n-1) # шаг
# МЕТОД РУНГЕ-КУТТА
ypk=np.zeros(n);ypk[0]=p[1] # значение функции в точке x[0]
# В цикле формируем еще n-1 значение функции
for i in range(n-1):
  k1=h*f(x[i],s);k2=h*f(x[i]+h/2,s+k1/2)
  k3=h*f(x[i]+h/2,s+k2/2);k4=h*f(x[i]+h,s+k3)
  t=(k1+2*k2+2*k3+k4)/6;s+=t;# значение функции в точке x[i+1]
  ypk[i+1]=s;
  # Метод Эйлера
yei=np.zeros(n);
s1=p[1];yei[0]=p[1] # значение функции в точке x[0]
# В цикле формируем еще n-1 значение функции
for i in range(n-1):
```

```
  t=h*f(x[i],s1);  s1+=t;yei[i+1]=s1;
yt=-np.sin(x)/2-np.cos(x)/2+5/2*np.exp(x)
print("Максимальная погрешность метода Эйлера равна %.8f"
      % np.max(abs(yt-yei)))
print("Максимальная погрешность метода Рунге-Кутта равна %.8f"
      % np.max(abs(yt-ypk)))
plt.plot(x,ypk,'--r*',x,yt,'--k',x,yei,'--og',lw=2,ms=7)
plt.legend(['Метод Рунге-Кутта','Точное решение',
            'Метод Эйлера'])
plt.grid(); plt.show()
```

Результатами работы программы являются значения максимальной по модулю погрешностью обоих методов и графическое решение задачи на интервале [0, 3] с шагом 0.1:

```
Максимальная погрешность метода Эйлера равна 9.92519172
Максимальная погрешность метода Рунге-Кутта равна 0.00060313
```

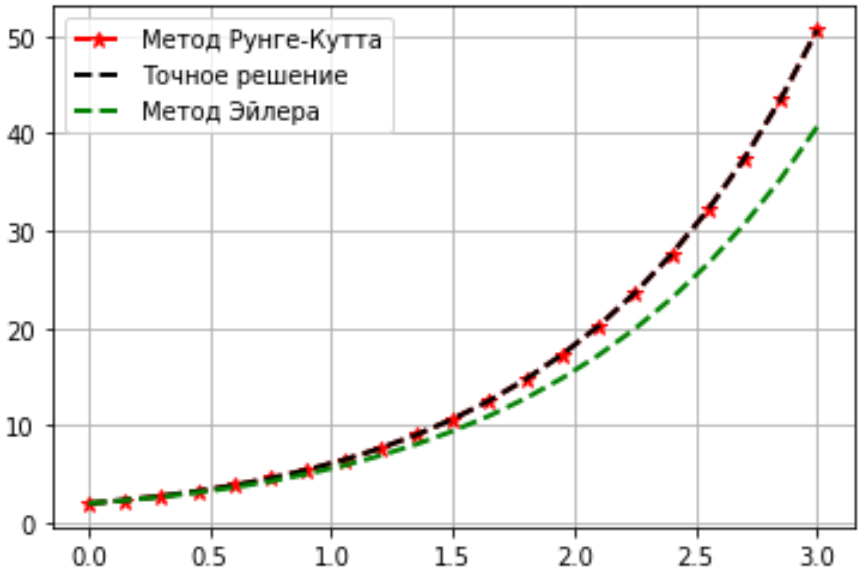


Рис. 8.1. Точное и численное решение задачи Коши для ОДУ первого порядка

Погрешность можно уменьшить, увеличив количество точек *n*.

8.2. Решение задачи Коши для систем дифференциальных уравнения первого порядка

Для численного решения систем дифференциальных уравнений также можно использовать методы Эйлера и Рунге-Кутта[1]. Покажем, как можно численно решить задачу Коши для системы двух дифференциальных уравнений, на примере.

Пример 8.2. Решить задачу Коши для системы двух дифференциальных уравнений первого порядка методом Эйлера. Сделать три шага. Взять шаг равным 0.1. Сравнить полученное решение с точным решением системы дифференциальных уравнений.

$$\begin{cases} y' = z - 5\cos x \\ z' = 2y + z \\ y(0) = 4 \\ z(0) = 7 \end{cases}.$$

Точное решение системы:

$$\begin{cases} y(x) = 2e^{-x} + 3e^{2x} - 2\sin x - \cos x, \\ z(x) = -2e^{-x} + 6e^{2x} + \sin x + 3\cos x \end{cases}.$$

Решение. Запишем формулы метода Эйлера для поиска y_{i+1} и z_{i+1}:

$$\begin{cases} y_{i+1} = y_i + h(z_i - 5\cos x_i) \\ z_{i+1} = z_i + h(2y_i + z_i) \end{cases}.$$ Вычислим первую пару значений:

$$\begin{cases} y_1 = y_0 + h(z_0 - 5\cos x_0) \\ z_1 = z_0 + h(2y_0 + z_0) \end{cases}.$$ x_0=0, y_0=4, z_0=7, h=0.1. Тогда

$$\begin{cases} y_1 = 4 + 0.1(7 - 5\cos 0) = 4.2. \\ z_1 = 7 + 0.1(2 \cdot 4 + 7) = 8.5 \end{cases}, \begin{cases} y_2 = y_1 + h(z_1 - 5\cos x_1) \\ z_2 = z_1 + h(2y_1 + z_1) \end{cases}, x_1 = x_0 + h.$$

$$\begin{cases} y_2 = 4.2 + 0.1(8.5 - 5\cos 0.1) = 4.5524979 \\ z_2 = 8.5 + 0.1(2 \cdot 4.2 + 8.5) = 10.260500 \end{cases}. \begin{cases} y_3 = y_2 + h(z_2 - 5\cos x_2) \\ z_3 = z_2 + h(2y_2 + z_2) \end{cases},$$

x_2=x_1+h; $$\begin{cases} y_3 = 4.5524979 + 0.1(10.2605 - 5\cos 0.2) = 5.0885146 \\ z_3 = 10.2605 + 0.1(2 \cdot 4.5524979 + 10.2605) = 12.304252 \end{cases}.$$

Результаты запишем в виде таблицы:

[1] Углубленный курс информатики. Лекция 11. Численные методы решения систем дифференциальных уравнений. – URL: https://portal.tpu.ru/SHARED/d/DOLGANOVIM/teaching/Tab4/Lec-11.pdf

x	y (метод Эйлера)	y точное	z (метод Эйлера)	z точное
0	4	4	7	7
0.1	4,2	4.27921211	8,5	8.60358763
0.2	4.5524979	4.73553036	10.2605	10.45235574
0.3	5.0885146	5.40161594	12.304252	12.61260604

Видим, что по мере удаления от начальных точек погрешность вычислений возрастает.

8.3. Нахождение точного решения дифференциальных уравнений и систем. Работа с библиотекой SymPy

Функция dsolve() библиотеки SymPy позволяет (но далеко не всегда) получить общее решение обыкновенного дифференциального уравнения *n*-го порядка. Синтаксис этой функции:

dsolve(eq, func=None, hint='default', simplify=True, ics=None, xi=None, eta=None, x0=0, n=6, **kwargs)

Решим пример 8.1 с помощью этой функции. Сначала найдем общее решение дифференциального уравнения.

Пример 8.3. Найти общее решение ОДУ первого порядка $y' = \sin(x) + y$.

Решение.

Первое, что нужно сделать – это создать функцию – экземпляр класса Function. Это можно сделать разными способами, например, передав cls=Function в функцию symbols: symbols('f', cls=Function). Можно это сделать и так: f=Function('f'). Набрав и выполнив код

from sympy import Function; f=Function('f');type(f)

получим: sympy.core.function.UndefinedFunction, то есть f принадлежит к классу неопределенных функций. Далее создаем объект Eq, соответствующий нашему дифференциальному уравнению t=Eq(f(x).diff(x,1)-f(x)-sin(x),0):

```
from sympy import *
x = symbols('x'); f = Function('f');t = Eq(f(x).diff(x,1)- f(x)-sin(x),0); t
```

Если теперь вывести на экран значение t, получим математическую запись нашего дифференциального уравнения:

$$-f(x) - \sin(x) + \frac{d}{dx} f(x) = 0$$

В формуле вместо переменной *y* стоит *f*(*x*). Теперь нужно применить функцию dsolve() для решения сформированного дифференциального

уравнения. Результатом работы функции будет объект типа Eq. Окончательно код выглядит так:

```
from sympy import dsolve, symbols, Function, Eq, sin
x= symbols('x');f=Function('f')
t=Eq(f(x).diff(x,1)-f(x)- sin(x),0);dsolve(t,f(x))
```

Выполнив код, получим решение дифференциального уравнения, содержащее одну константу C_1:

$$f(x) = C_1 e^x - \frac{\sin(x)}{2} - \frac{\cos(x)}{2}$$

Покажем теперь, как можно решить задачу Коши для нашего дифференциального уравнения, задав, начальное условие: $y(0)=2$.

Из всего множества решений нам нужно так подобрать C_1, чтобы в точке $x=0$ выполнялось начальное условие $y(0)=2$. Сведем задачу к решению линейного уравнения с одним неизвестными: $f(x)=0$, где $f(x)$ – полученное решение. Код решения задачи Коши для примера 8.3:

```
from sympy import dsolve, symbols, Function, Eq, sin, solve
x= symbols('x');f=Function('f')
t=Eq(f(x).diff(x,1)-f(x)- sin(x),0)
y=dsolve(t,f(x))
y0=y.rhs.subs({x:0})-2 # значение функции в точке 0
C1=symbols(" C1")
const=solve([y0],[C1]);const
```

Результат: {C1: 5/2}

Точное решение можно вывести на экран, добавив к коду строки:

```
w=y.subs(const)
w
```

Выполнив код, получим:

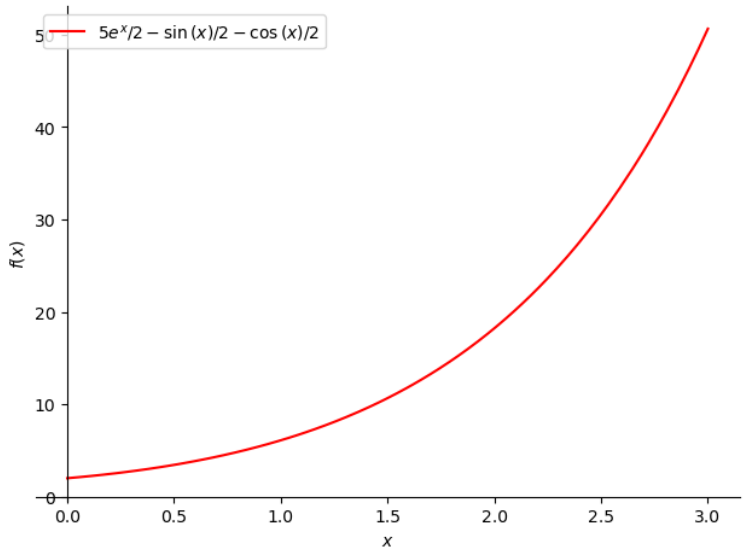
$$f(x) = \frac{5e^x}{2} - \frac{\sin(x)}{2} - \frac{\cos(x)}{2}$$

Комментарий. y.rhs (right hand side) возвращает правую часть нашего уравнения $C_1 e^x - \frac{\sin x}{2} - \frac{\cos x}{2}$. Она должна равняться 2. Из этого уравнения с помощью функции *solve*() ищется константа C1.

Для построения графика после решения задачи Коши надо выполнить строки:

```
from sympy import plot
plot(y.rhs.subs(const),(x,0,3), line_color='red', legend=True)
```

Выполнив код, получим:



Покажем, как можно решить задачу Коши для дифференциального уравнения второго порядка.

Пример 8.4. Найти общее решение ОДУ второго порядка и решение задачи Коши $\begin{cases} y'' - 2y' + 10y = 0 \\ y(0) = 3,\ y'(0) = 1. \end{cases}$

Решение.

Первое, что нужно сделать – это создать функцию – экземпляр класса Function. Набрав и выполнив код:

```
from sympy import Function, diff, Eq, symbols
f=Function('f')
x= symbols('x')
t=Eq(f(x).diff(x,2)-2*f(x).diff(x)+10*f(x),0)
t
```

получим запись нашего дифуравнения:

$$10f(x) - 2\frac{d}{dx}f(x) + \frac{d^2}{dx^2}f(x) = 0$$

Теперь нужно применить функцию dsolve() для решения сформированного дифференциального уравнения. Дописываем строки:

```
from sympy import dsolve
y=dsolve(t,f(x)); y
```

и выполняем код. Получаем общее решение дифференциального уравнения, содержащего 2 константы. Вот получившееся решение дифференциального уравнения: $f(x) = \left(C_1 \sin(3x) + C_2 \cos(3x)\right)e^x$

Добавляем к коду строки:

```
y0=y.rhs.subs({x:0})-3 # значение функции в точке 0
y01=diff(y.rhs,x).subs({x:0})-1 # значение производной в точке 0
```

C1,C2=symbols(" C1,C2")
sol_const=solve([y0,y01],[C1,C2])
sol_const
Решение: {C1: -2/3, C2: 3}

Ответ: результатом решения задачи Коши для данного уравнения является функция $\left(-\frac{2}{3}\sin 3x+3\cos 3x\right)e^{x}$.

Покажем, как можно решить краевую задачу, когда условия заданы на концах отрезка.

Пример 8.5. Решить краевую задачу $\begin{cases} y''-5y'+4y=8 \\ y(0)=1 \\ y(\ln 2)=7 \end{cases}$.

Код программы:

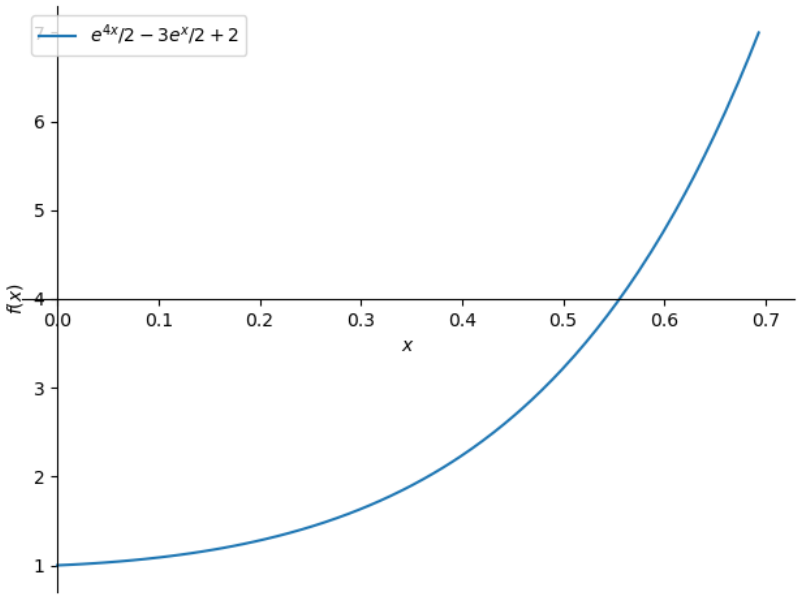
```
from sympy import *
x= symbols('x')
f=Function('f')
t=Eq(f(x).diff(x,2)-5*f(x).diff(x)+4*f(x),8)
y=dsolve(t,f(x));t=y.rhs
y0=t.subs({x:0})-1;
y01=t.subs({x:log(2)})-7
C1,C2=symbols(" C1,C2")
sol_const=solve([y0,y01],[C1,C2])
w=t.subs(sol_const)
plot(w,(x,0,log(2)),legend=True)
w
```

Выполнив код, получим решение и график:

Решение задачи:

$$\frac{e^{4x}}{2}-\frac{3e^{x}}{2}+2$$

Графическое решение:



Покажем, как можно решить задачу Коши для системы двух дифференциальных уравнений первого порядка на примере.

Пример 8.6. Решить задачу Коши $\begin{cases} \dfrac{dx}{dt} = -2x(t) + 4y(t) \\ \dfrac{dy}{dt} = -x(t) + 3y(t) \\ x(0) = 3 \\ y(0) = 0 \end{cases}$.

Вот код программы.

```
from sympy import *
t=symbols('t')
y1=Function('y1')
y2=Function('y2')
eqs=(Eq(y1(t).diff(t),-2*y1(t)+4*y2(t)),
    Eq(y2(t).diff(t),-y1(t)+3*y2(t)))
s=dsolve(eqs)       # Общее решение
display(s)
y1g=s[0].args[1] # или y1g=s[0].rhs
y2g=s[1].args[1] # y2g=s[1].rhs
# Находим C1 и C2
sol=solve([y1g.subs(t,0)-3,y2g.subs(t,0)-0])
y1=y1g.subs(sol); y2=y2g.subs(sol)
y11=simplify(y1); y12=simplify(y2)
display(y11, y12); display (y1, y2)
```

В результате на экран будут выведены общее решение и решение задачи Коши:

```
[Eq(y1(t), 4*C1*exp(-t) + C2*exp(2*t)), Eq(y2(t), C1*exp(-t) + C2*exp(2*t))]
```

$$\left(4 - e^{3t}\right) e^{-t}$$

$$\left(1 - e^{3t}\right) e^{-t}$$

$$-e^{2t} + 4e^{-t}$$

$$-e^{2t} + e^{-t}$$

Комментарий к программе. Объявляем переменную *t* символьной, а переменные y1 (x(t)) и y2 (y(t)) функциями. Определяем два уравнения. Eq(y1(t).diff(t),-2*y1(t)+4*y2(t) эквивалентно первому уравнению $\frac{dx}{dt} = -2x(t) + 4y(t)$, Eq(y2(t).diff(t),-y1(t)+3*y2(t)) эквивалентно второму уравнению $\frac{dy}{dt} = -x(t) + 3y(t)$. Переменная *eqs* задает систему уравнений, которую решаем с помощью функции dsolve(). Если вывести на экран s, получим общее решение системы дифференциальных уравнений. [Eq(y1(t), 4*C1*exp(-t) + C2*exp(2*t)), Eq(y2(t), C1*exp(-t) + C2*exp(2*t))]. s[0] – это найденное значение y1(t): $y_1(t) = 4C_1e^{-t} + C_2e^{2t}$ Переменная y1g – это правая часть этого уравнения, а именно $4C_1e^{-t} + C_2e^{2t}$. Если вывести н экран s[0].args[0], то получим левую часть уравнения: $y_1(t)$. Аналогично, s[1] – это найденное значение y2(t): $y_2(t) = C_1e^{-t} + C_2e^{2t}$, y2g – это правая часть уравнения: $C_1e^{-t} + C_2e^{2t}$. Решение задачи Коши находим из решения системы уравнений $\begin{cases} y1g(0) = 3 \\ y2g(0) = 0 \end{cases}$. Вывод на экран решения системы (sol) дает значения C1 и C2: {C1: 1, C2: -1}. Вывод y11 и y12 дает решение задачи Коши: $(4-e^{3t})e^{-t}$ – это *x(t)*, а $(1-e^{3t})e^{-t}$ – *y(t)*. Если не использовать функции simplify, то решение получится в виде: $-e^{2t}+4e^{-t}$ и $-e^{2t}+e^{-t}$.

8.4. Нахождение численного решения дифференциальных уравнений и систем. Работа с библиотекой SciPy

Для численного решения дифференциальных уравнений и систем дифференциальных уравнений можно воспользоваться библиотекой SciPy. Для решения ОДУ в библиотеке SciPy есть несколько функций. Среди них – odeint() и solve_ivp(). Рассмотрим первую из них.

Синтаксис функции odeint():

odeint(func, y0, t, args=(), Dfun=None, col_deriv=0, full_output=0, ml=None, mu=None, rtol=None, atol=None, tcrit=None, h0=0.0, hmax=0.0, hmin=0.0, ixpr=0, mxstep=0, mxhnil=0, mxordn=12, mxords=5, printmessg=0, tfirst=False)

Обязательными аргументами являются первые три:

func – функция, задающая правую часть дифференциального уравнения $\frac{dy}{dt} = func(y,t,\ldots)$ первого порядка или правые части системы таких уравнений. Если порядок следования аргументов в функции $func(t,y,\ldots)$, значение параметра tfirst должно быть True;

y0 – вектор начальных условий;

t – последовательность. Точки интервала, на котором необходимо получить решение. Начальная точка должна быть первым элементом этой последовательности. Последовательность должна быть монотонно возрастающей или монотонно убывающей. Допускаются повторяющиеся значения.

Пример 8.7. Решить задачу Коши для ОДУ первого порядка $\begin{cases} y' = \sin x + y \\ y(0) = 2 \end{cases}$. Решение получить в точках отрезка [0, 3] с шагом 0.01, используя функцию odeint(). Нанести на график полученное решение и известное точное решение задачи $y = \frac{5}{2}e^x - \frac{\sin x}{2} - \frac{\cos x}{2}$. Вычислить максимум модуля разности между точным и приближенным значениями в 301 точке отрезка [0, 3].

Решение:

```
import numpy as np; import matplotlib.pyplot as plt
from scipy.integrate import odeint
def func(y,x):    dydx=np.sin(x)+y;    return dydx
x2 = np.linspace(0,3,301) # интервал поиска решения
y0 = [2] # Значения функции в точке x2[0]
```

```
y = odeint(func, y0, x2);
yt=-np.sin(x2)/2-np.cos(x2)/2+5/2*np.exp(x2)
plt.plot(x2,y,':r*',x2,yt,'--g',lw=3,ms=4)
plt.legend(['Численное решение','Точное решение'])
p=np.max(abs((yt-y.T)))
print("Максимальная погрешность метода odeint равна %.8f" % p)
plt.show()
```

Результаты решения:

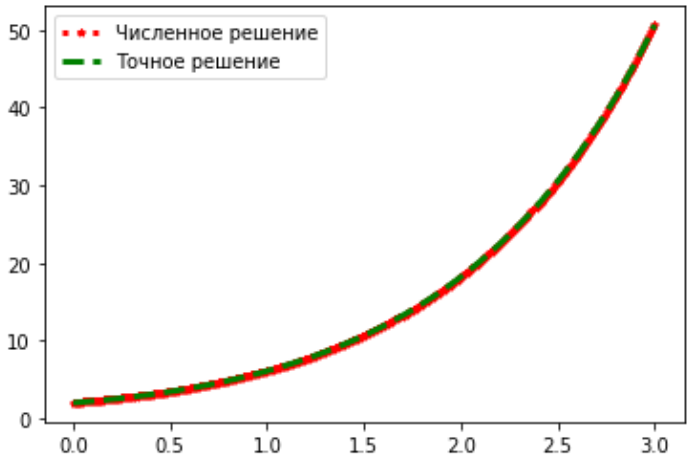


Рис. 8.2. Точное и приближенное решение задачи Коши

Результатом работы программы является также и значение максимальной абсолютной погрешности:

Максимальная погрешность метода odeint равна 0.00000368

Оба графика практически неразличимы: точное решение задачи практически совпало с приближенным, что подтверждает и найденное значение максимума модуля разности между точным и приближенным решениями.

При необходимости получить числовые значения решения в приведенной программе нужно вывести на экран значения массива *y*.

Рассмотрим еще одну функцию для решения ОДУ – функцию solve_ivp (). Ее синтаксис:

solve_ivp (fun , t_span , y0 , method = 'RK45' , t_eval = None , visible_output = False , events = None , vectorized = False , args = None , ** options)

С помощью этой функции можно численно решить задачу Коши для как для ОДУ первого порядка, так и для системы обыкновенных дифференциальных уравнений.

Назначение некоторых параметров функции:

fun – правые части системы дифуравнений. Вызываемая функция задается в виде fun(t, y);

t_span – кортеж из двух чисел, являющихся началом и концом интервала интегрирования;

y0 – массив начальных значений. Размерность равна количеству уравнений в системе;

method – метод решения задачи. Может быть одним из следующих: 'RK45' – метод Рунге-Кутта порядка 4, 'RK23' – аналогичный метод порядка 2; 'DOP853' – метод Рунге-Кутта порядка 8; 'Radau' – неявный метод Рунге-Кутта семейства Радау, метод порядка 5; 'BDF' – неявный многошаговый метод переменного порядка (от 1 до 5), основанный на формуле обратного дифференцирования для аппроксимации производной; 'LSODA' – метод Адамса/BDF с автоматическим определением жесткости [1];

t_eval – точки, в которых нужно вычислить значение функции. Он должны быть отсортированы и лежать в пределах *t_span*. Если значение равно None (по умолчанию), используются точки, выбранные решателем.

С остальными параметрами можно ознакомиться в документации по функции.

Среди выходных параметров основными являются:

t – точки, в которых получено решение;

y – решение задачи в точках t.

Приведем решение примера 8.6 с использованием функции solve_ivp ():

```
import numpy as np; import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
def func(t,y):
   dydt=np.sin(t)+y;
   return dydt
x2 = np.linspace(0,3,301) # интервал поиска решения
y1 = [2] # Значения функции  в точке x2[0]
y = solve_ivp(func,t_span =(0,3),y0=y1,t_eval=x2)
yt=-np.sin(x2)/2-np.cos(x2)/2+5/2*np.exp(x2)
plt.plot(x2,y.y.T,':r*',x2,yt,'--g',lw=3,ms=4)
plt.legend(['Численное решение','Точное решение'])
p=np.max(abs((yt-y.y)))
print("Максимальная погрешность метода solve_ivp равна %.8f" % p)
```

plt.show()

Краткий комментарий к коду. Во избежание путаницы вместо переменной *x* использовалась переменная *t*. Результат работы программы – y.y. Для приведения y.y и x2 к одной размерности использовался метод .T.

Результаты решения задачи:

Графическое решение:

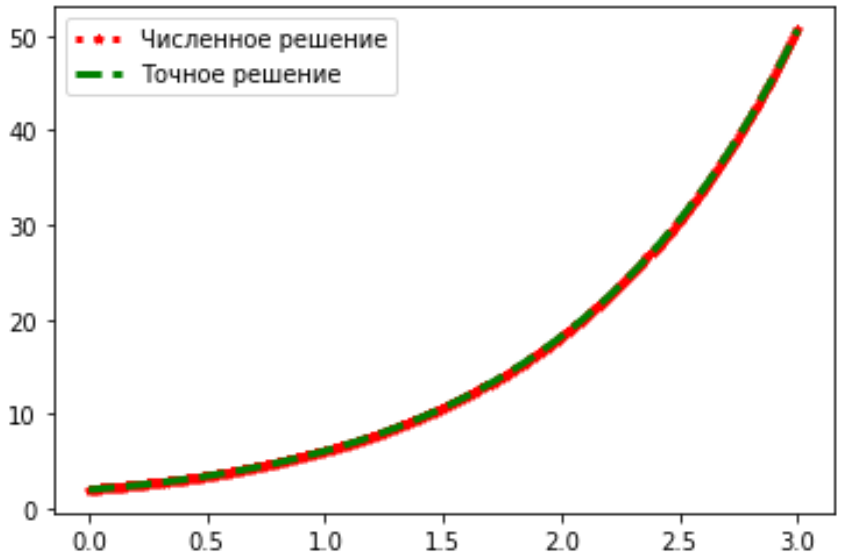


Рис. 8.3. Решение задачи методом solve_ivp

Максимальная погрешность метода solve_ivp равна 0.00306482

Видим, что погрешность решения с автоматическим выбором метода решения больше, чем при решении методом odeint().

Решим теперь систему дифференциальных уравнений с помощью функций odeint() и solve_ivp().

Пример 8.8. Решить задачу Коши для системы дифуравнений.

$$\begin{cases} y' = z - 5\cos x \\ z' = 2y + z \\ y(0) = 4 \\ z(0) = 7 \end{cases}$$

. Решение получить в точках отрезка [0, 2]. Вычислить 50 значений функций *y* и *z*.

Решение с использованием функции odeint():

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
def F(s,x):
```

```
  dydx=s[1]-5*np.cos(x)
  dzdx=2*s[0]+s[1]
  return [dydx, dzdx]
s0=[4,7] # список - значения функций в нулевой точке
x = np.linspace(0, 2, 50);
s=odeint(F,s0,x) # ссылка на двумерный массив решения
plt.plot(x,s[:,0],'r',linewidth=3,label="y(x)")
plt.plot(x,s[:,1],'g--',linewidth=3,label="z(x)")
yt=2*np.exp(-x)+3*np.exp(2*x)-2*np.sin(x)-np.cos(x)
zt=-2*np.exp(-x)+6*np.exp(2*x)+np.sin(x)+3*np.cos(x)
plt.plot(x,yt,'k-',linewidth=2,label="yt")
plt.plot(x,zt,'m-*',linewidth=2,label="zt")
plt.xlabel("x")
plt.ylabel("y(x),     z(x)")
plt.legend()
plt.grid()
plt.show()
er1=np.max(abs(s[:,0]-yt))
er2=np.max(abs(s[:,1]-zt))
er=max(er1,er2)
print("Максимальная погрешность метода odeint равна %.8f" % er)
```

В коде приняты обозначения: s[0] – это *y*, s[1] – это *z*.

Результат работы программы:

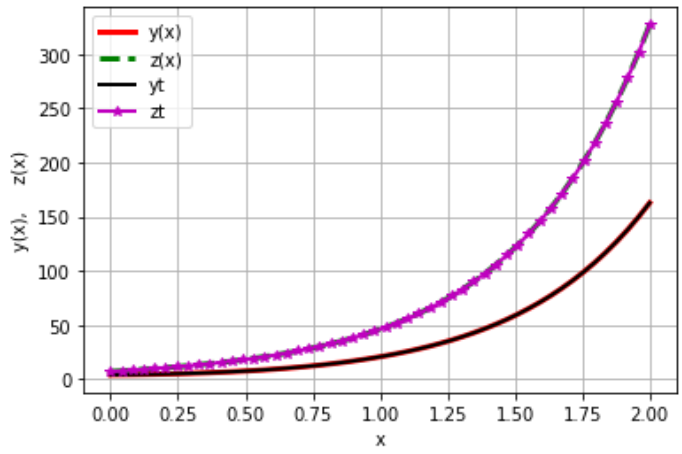


Рис. 8.4. Точное и приближенное решения задачи Коши для системы двух дифференциальных уравнений

Максимальная погрешность метода odeint равна 0.00001819

И в этом примере численное решение практически совпало с точным. Значения *y* (s[:,0]) и *z* (s[:,1]) при необходимости можно вывести на экран.

Решение с помощью функции solve_ivp():

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
def func(t,y):
    return [y[1]-5*np.cos(t),2*y[0]+y[1]]
    # t - это x
    # y[0] - это y
    # y[1] - это z
s0=[4,7] # список - значения функций в нулевой точке
t = np.linspace(0,2,50); # точки для вычисления функций
sol = solve_ivp(func,t_span=(0,2),y0=s0,t_eval=t)
plt.plot(t,sol.y[0],'r',linewidth=3,label="y(x)")
plt.plot(t,sol.y[1],'g--',linewidth=3,label="z(x)")
yt=2*np.exp(-t)+3*np.exp(2*t)-2*np.sin(t)-np.cos(t)
zt=-2*np.exp(-t)+6*np.exp(2*t)+np.sin(t)+3*np.cos(t)
plt.plot(t,yt,'k-',linewidth=2,label="y точное")
plt.plot(t,zt,'m-*',linewidth=2,label="z точное")
plt.xlabel("x")
plt.ylabel("y(x),     z(x)")
plt.legend()
plt.grid()
plt.show()
er1=np.max(abs(sol.y[1]-zt))
er2=np.max(abs(sol.y[0]-yt))
er=max(er1,er2)
print("Максимальная погрешность метода solve_ivp равна %.8f" % er)
```

Результат:

Максимальная погрешность метода solve_ivp равна 0.09790382

Графическое решение представлено на рис. 6.5:

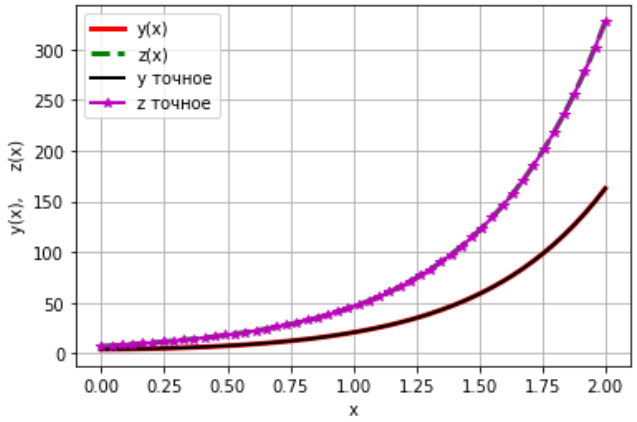


Рис. 8.5. Решение задачи Коши для системы двух дифференциальных уравнений методом solve_ivp()

И в этом случае погрешность получилась больше, чем при решении методом odeint().

Решение задачи Коши для дифференциальных уравнений высших порядков может быть сведено к решению задачи Коши для системы дифференциальных уравнений первого порядка. Покажем, как это можно сделать, на примере.

Пример 8.9. Решить задачу Коши для дифференциального уравнения второго порядка:

$$\begin{cases} y'' - 2y' + 10y = e^{-x} \\ y(1) = 2 \\ y'(1) = 0.7 \end{cases}.$$

Решение найти с помощью функции odeint(). Построить график приближенного решения на отрезке [1, 3.5].

Решение. Сведем задачу решения дифференциального уравнения второго порядка к решению системы двух дифференциальных уравнений первого порядка: $\begin{cases} y'(x) = y_1(x) \\ y''(x) = y_1'(x) = 2y_1(x) - 10y(x) + e^{-x} \end{cases}$. При этом начальные условия перепишутся в виде $\begin{cases} y(1) = 2 \\ y_1(1) = 0.7 \end{cases}$.

Решение системы двух дифуравнений первого порядка методом Эйлера было рассмотрено выше.

Точное решение задачи в данном случае можно найти с помощью функции dsolve() библиотеки SymPy.

Решим полученную систему численно с помощью функции odeint().

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
def func(t,y):
    y0, y1 = y
# y0 - это решение дифуравнения, y1 - первая производная
# Возвращаются правые части системы дифуравнений первого порядка
    dydt = [y1, 2*y1-10*y0+np.exp(-t)]
    return dydt
x2 = np.linspace(1,3.5,30) # интервал поиска решения
y0 = [2.0, .7] # Значения функции и производной в точке x2[0]
y = odeint(func, y0, x2, tfirst=True)
# y[:,0] - численное решение дифуравнения в точках x2
# y[:,1] - вычисленное значение первой производной в точках x2
plt.plot(x2,y[:,0],':r*',lw=3,ms=10)
plt.legend(['Численное решение'])
plt.show()
```

Графическое решение примера 8.9 представлено на рис. 8.6:

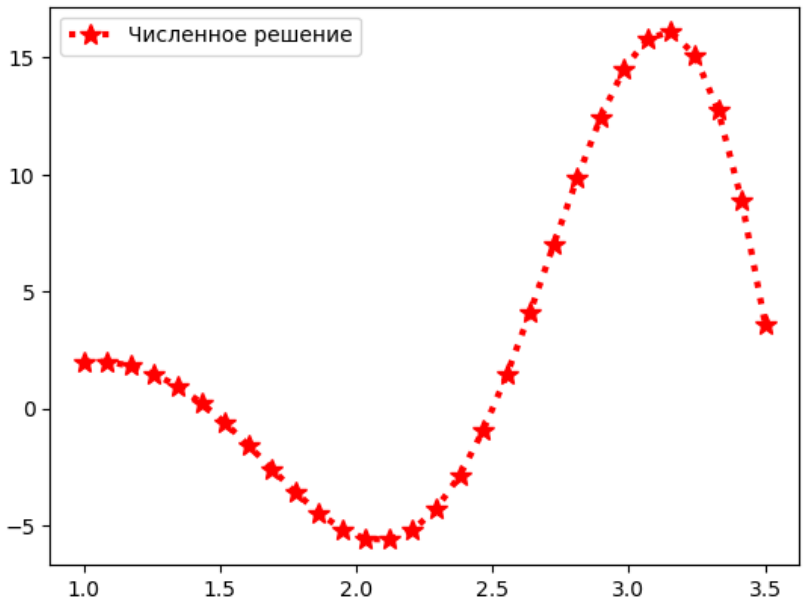


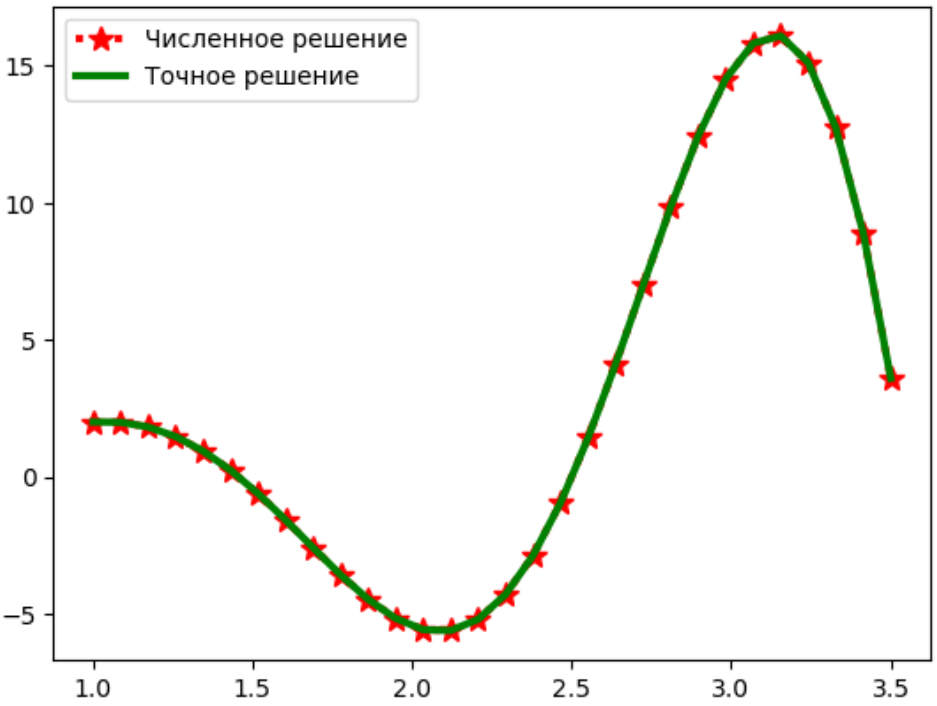
Рис. 8.6. Численное решение задачи Коши для дифференциального уравнения второго порядка

Комментарий к коду. Подгружаем нужные библиотеки и функции. Находим численное решение дифференциального уравнения. Для этого создаем функцию func(t,y) с правыми частями системы дифференциальных уравнений первого порядка. В коде *y0* – это $y(x)$, *y1* – это $y_1(x)$. Задаем интервал поиска решения, начальные условия и решаем задачу с помощью функции odeint(), передав в нее необходимые параметры. Решение задачи – вектор y[:,0]. Наносим на график приближенное решение.

Так можно найти точное и приближенное решение задачи 8.9:

```
from sympy import *
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
x= symbols('x');f=Function('f')
eq=Eq(f(x).diff(x,2)-2*f(x).diff(x)+10*f(x),exp(-x));
yy=dsolve(eq,f(x));u=yy.rhs
yy0=u.subs({x:1})-2;# значение функции в точке 1
yy01=diff(u,x).subs({x:1})-0.7 # значение производной в точке 1
C1,C2=symbols(" C1,C2")
sol_const=solve([yy0,yy01],[C1,C2])
w=yy.subs(sol_const);w1=w.rhs
def func(t,y):
    y0, y1 = y
# y0 - это решение дифуравнения, y1 - первая производная
# Возвращаются правые части системы дифуравнений первого порядка
    dydt = [y1, 2*y1-10*y0+np.exp(-t)]
    return dydt
x2 = np.linspace(1,3.5,30) # интервал поиска решения
y0 = [2.0, .7] # Значения функции и производной в точке x2[0]
y = odeint(func, y0, x2, tfirst=True)
# y[:,0] - численное решение дифуравнения в точках x2
# y[:,1] - вычисленное значение первой производной в точках x2
ff=lambdify(x,w1)
plt.plot(x2,y[:,0],':r*',x2,ff(x2),'-g',lw=3,ms=10)
plt.legend(['Численное решение', 'Точное решение'])
plt.show()
```

Результат:



Данную задачу можно решить и с использованием функции solve_ivp ().