Южный федеральный университет

Н. М. Полякова, Е. В. Ширяева

Python 3. Создание графического интерфейса пользователя

(на примере решения методом пристрелки краевой задачи для линейных обыкновенных дифференциальных уравнений)

Ростов-на-Дону
2017

Содержание

Введение

В пособии описано создание на языке Python 3.5 графического интерфейса — виджет ввода данных и визуализация результатов расчета — на примере реализации программы решения краевой задачи для системы линейных обыкновенных дифференциальных уравнений при помощи метода пристрелки. Демонстрируются широкие возможности, которые предоставляет язык Python, точнее его версия Python 3.5, для проведения студентами, магистрами, аспирантами научной работы при решении различных математических задач. Упоминание про конкретную версию Python 3.5 не является случайным. Дело в том, что версии Python 2+ и версии Python 3+ различаются между собой некоторыми деталями, как впрочем, и различные версии Python 3+.

Структура языка Python позволяет достаточно просто использовать большое количество модулей для создания интерфейса программ (например, модуль **tkinter**), для численного и аналитического решения математических задач (например, модули **numpy**, **scipy**), а также для анализа, в частности, визуализации, результатов вычислений (например, модуль **matplotlib**). Именно на основе указанных модулей реализована программа метода пристрелки. Программа не является универсальной, а решается вполне конкретная краевая задача для трех линейных неоднородных дифференциальных уравнений. Внимательное изучение приводимого кода программы позволяет легко модифицировать код, приспособив его для решения аналогичных задач.

Заметим, что код программы не является оптимальным и содержит много излишеств и «украшательств». Основная цель, которая преследовалась при создании программы — создание простого, достаточно прозрачного для понимания кода. Поняв основные принципы построения программы, в дальнейшем несложно превратить программу в оптимальную и универсальную.

Опишем структуру пособия. В п. 1 дана общая постановка краевой задачи для системы обыкновенных дифференциальных уравнений и рассмотрен конкретный пример, решение которого в дальнейшем реализует программа. Реализации программы дана в п. 2. В п. 3 демонстрируются некоторые возможности использования Python с системой компьютерной верстки LaTeX. Это особенно важно при создании отчетов о проделанной работе. Наконец, в п. 4 приведен ряд контрольных заданий.

Работа поддержана Базовой частью государственного задания Министерства образования и науки РФ № 1.5169.2017/8.9.

1 Постановка задачи

Краевая задача для системы обыкновенных неоднородных линейных диференциальных уравнений в общем случае имеет вид (см., например, [1, 2])

$$\frac{dy_i(x)}{dx} = \sum_{k=1}^{n} A_{ik}(x)y_k(x) + f_i(x), \quad i = 1, \ldots, n, \quad a < x < b, \tag{1.1}$$

$$\sum_{k=1}^{n} B_{ik}y_k(a) = b_i, \quad i = 1, \ldots, n - r, \tag{1.2}$$

$$\sum_{k=1}^{n} C_{ik}y_k(b) = c_i, \quad i = 1 \ldots, r. \tag{1.3}$$

Здесь

$y_i(x)$ — неизвестные функции (вектор размерности n),

$A_{ik}(x)$ — известная матрица размерности $n \times n$,

B_{ik} — известная матрица размерности $(n - r) \times n$,

C_{ik} — известная матрица размерности $r \times n$,

$f_i(x)$ — известные функции (вектор размерности n),

b_i — известный вектор (размерности $n - r$),

c_i — известный вектор (размерности r).

Матричная форма записи задачи (1.1)–(1.3) будет

$$\frac{d\boldsymbol{y}(x)}{dx} = \boldsymbol{A}(x)\boldsymbol{y}(x) + \boldsymbol{f}(x), \quad a < x < b, \tag{1.4}$$

$$\boldsymbol{B}\boldsymbol{y}(a) = \boldsymbol{b}, \tag{1.5}$$

$$\boldsymbol{C}\boldsymbol{y}(b) = \boldsymbol{c}. \tag{1.6}$$

Обратим внимание на то, что матрицы $\boldsymbol{B}$, $\boldsymbol{C}$ являются прямоугольными. Считаем, что ранг матрицы $\boldsymbol{B}$ равен $(n - r)$, а ранг матрицы $\boldsymbol{C}$ равен r

$$\operatorname{rank} \boldsymbol{B} = n - r, \quad \operatorname{rank} \boldsymbol{C} = r. \tag{1.7}$$

1.1 Общая схема решения задачи методом пристрелки

Подробно описанный способ решения задачи (1.1)–(1.3) имеется, например, в [1, с. 565, 571, 572]. Основная идея метода пристрелки заключается в замене решения краевой задачи решением задачи Коши. При этом недостающие начальные условия для задачи Коши, например, при $x = a$ определяются таким образом, чтобы удовлетворить краевым условиям при $x = b$. Конечно, можно задавать недостающие значения для задачи Коши и при $x = b$, определяя их так, чтобы удовлетворить краевым условиям при $x = a$. В первом случае задача Коши решается, начиная с точки $x = a$ (слева направо), а во втором, начиная с точки $x = b$ (справа налево). Более того, недостающие начальные условия можно задавать одновременно как на левом конце, так и на правом, выполняя решение задачи Коши слева направо и справа налево с последующим сшиванием решения в какой либо промежуточной точке отрезка $[a, b]$.

Приведем лишь краткую схему решения задачи для случая, когда решение задачи Коши выполняется от точки $x = a$ до точки $x = b$ (слева направо).

Рассмотрим алгебраическую систему линейных уравнений (1.2)

$$\sum_{k=1}^{n} B_{ik} y_k(a) = b_i, \quad i = 1, \ldots, n - r \tag{1.8}$$

или в матричной форме

$$\boldsymbol{B}\boldsymbol{y}(a) = \boldsymbol{b}. \tag{1.9}$$

Так как ранг матрицы B_{ik} равен $(n-r)$, общее решение системы (1.8) имеет вид

$$y_k(a) = \varphi_k^0 + \sum_{s=1}^{r} \theta_s \varphi_k^s, \quad k = 1, \ldots, n \tag{1.10}$$

или в матричной форме

$$\boldsymbol{y}(a) = \boldsymbol{\varphi}^0 + \sum_{s=1}^{r} \theta_s \boldsymbol{\varphi}^s, \tag{1.11}$$

Здесь θ_s — произвольные константы.

Вектор φ_k^0 — произвольное решение неоднородной системы (1.8)

$$\sum_{k=1}^{n} B_{ik}\varphi_k^0 = b_i, \quad i = 1, \ldots, n-r \tag{1.12}$$

или в матричной форме

$$\boldsymbol{B}\boldsymbol{\varphi}^0 = \boldsymbol{b}. \tag{1.13}$$

Векторы φ_k^s — произвольные решения однородной системы (1.8)

$$\sum_{k=1}^{n} B_{ik}\varphi_k^s = 0, \quad i = 1, \ldots, n-r, \quad s = 1, \ldots, r \tag{1.14}$$

или в матричной форме

$$\boldsymbol{B}\boldsymbol{\varphi}^s = 0. \tag{1.15}$$

Далее считаем, что набор векторов $\boldsymbol{\varphi}^0$, $\boldsymbol{\varphi}^s$ $(s = 1, \ldots, n)$ известен.

1. Решаем задачу Коши для неоднородных уравнений (1.1)

$$\frac{dy_i(x)}{dx} = \sum_{k=1}^{n} A_{ik}(x)y_k(x) + f_i(x), \quad i = 1, \ldots, n, \quad a < x < b, \tag{1.16}$$

с неоднородными начальными условиями

$$y_i(a) = \varphi_i^0, \quad i = 1, \ldots, n. \tag{1.17}$$

Для решения задачи (1.16), (1.17) на интервале $a \leqslant x \leqslant b$ используем обозначение

$$y_i(x) = Y_i^0(x), \quad i = 1, \ldots, n, \quad a \leqslant x \leqslant b. \tag{1.18}$$

2. Решаем задачи Коши для однородных уравнений (1.1)

$$\frac{dy_i(x)}{dx} = \sum_{k=1}^{n} A_{ik}(x)y_k(x), \quad i = 1, \ldots, n, \quad a < x < b, \tag{1.19}$$

с неоднородными начальными условиями

$$y_i(a) = \varphi_i^s, \quad i = 1, \ldots, n, \quad s = 1, \ldots, r, \tag{1.20}$$

Для решения серии ($s = 1, \ldots, r$) задач (1.19), (1.20) на интервале $a \leqslant x \leqslant b$ используем обозначение

$$y_i(x) = Y_i^s(x), \quad i = 1, \ldots, n, \quad s = 1, \ldots, r, \quad a \leqslant x \leqslant b. \tag{1.21}$$

3. Таким образом, общее решение задачи Коши (1.1), (1.2) с учетом соотношения (1.10) может быть записано в виде

$$y_k(x) = Y_k^0(x) + \sum_{s=1}^{r} \theta_s Y_k^s(x), \quad k = 1, \ldots, n. \tag{1.22}$$

Действительно, непосредственной подстановкой, принимая во внимание обозначения (1.18), (1.21), можно убедиться, что соотношение (1.22) удовлетворяет уравнениям (1.1) и условиям (1.2).

Напомним, что параметры θ_s в формуле (1.22) являются произвольными. Для их определения используем условия (1.3). Подставляя (1.22) при $x = b$ в (1.3), получим

$$\sum_{k=1}^{n} C_{ik} \left(Y_k^0(b) + \sum_{s=1}^{r} \theta_s Y_k^s(b) \right) = c_i, \quad i = 1, \ldots, r. \tag{1.23}$$

Изменив порядок суммирования и перегруппировав члены, запишем (1.23) в форме

$$\sum_{s=1}^{r} \left(\sum_{k=1}^{n} C_{ik} Y_k^s(b) \right) \theta_s = c_i - \sum_{k=1}^{n} C_{ik} Y_k^0(b), \quad i = 1, \ldots, r. \tag{1.24}$$

Введем обозначения

$$M_{is} = \sum_{k=1}^{n} C_{ik} Y_k^s(b), \quad i, s = 1, \ldots, r, \tag{1.25}$$

$$m_i = c_i - \sum_{k=1}^{n} C_{ik} Y_k^0(b), \quad i = 1, \ldots, r. \tag{1.26}$$

Тогда (1.24) записывается в виде системы линейных алгебраических уравнений относительно неизвестных переменных θ_s

$$\sum_{k=1}^{n} M_{is} \theta_s = m_i, \quad i = 1, \ldots, r. \tag{1.27}$$

Решая (1.24), определим θ_s. Подставляя θ_s в (1.22), получим решение исходной задачи (1.1)–(1.3).

Замечание. Представление решения непосредственно в виде формулы (1.22) является эффективным с точки зрения скорости вычислений. Предполагается, что решения $Y_k^0(x)$, $Y_k^s(x)$ известны в каждой точке x (или некотором дискретном наборе точек x_j при численном решении). Однако для хранения решений в каждой точке требуется значительный объем памяти. Более того, при численной реализации метода, задачи Коши (1.16), (1.17) и (1.19), (1.20), как правило решаются методами, например Рунге–Кутты, с автоматическим выбором шага. В частности, это означает, что решения $Y_k^0(x)$ и $Y_k^s(x)$ $(s = 1, \ldots, r)$ определяются на различных дискретных наборах точек из интервала $[a, b]$. Это не позволяет использовать формулу (1.22) для представления решения задачи. Точнее, в этом случае следует либо использовать некоторые схемы интерполяции для согласования функций $Y_k^0(x)$ и $Y_k^s(x)$, либо использовать специальные методы Рунге–Кутты, позволяющие получать решения в фиксированных дискретных наборах точек.

Существует вариант метода пристрелки, позволяющий избежать указанных трудностей. Для этого заметим, что для построения матрицы M_{ik} и вектора m_i требуются значения функций $Y_k^0(x)$ и $Y_k^s(x)$ лишь в точке $x = b$, то есть $Y_k^0(b)$ и $Y_k^s(b)$. Таким образом, для определения коэффициентов θ_s нет необходимости при решении задач (1.16), (1.17) и (1.19), (1.20) сохранять значения функций $Y_k^0(x)$ и $Y_k^s(x)$ в каждой точке интервала $[a, b]$.

Для того чтобы получить решение краевой задачи (1.1)–(1.3), после того как определены параметры θ_s достаточно решить задачу Коши

$$\frac{dy_i(x)}{dx} = \sum_{k=1}^{n} A_{ik}(x) y_k(x) + f_i(x), \quad i = 1, \ldots, n, \quad a < x < b, \qquad (1.28)$$

с начальными условиями

$$y_k(a) = \varphi_k^0 + \sum_{s=1}^{r} \theta_s \varphi_k^s, \quad k = 1, \ldots, n. \tag{1.29}$$

Такой метод экономит память, так как нет необходимости хранить функции $Y_k^0(x)$ и $Y_k^s(x)$, и устраняет описанные выше недостатки по согласованию значений этих функций в отдельных точках x. При этом снижается скорость вычислений, так как требуется выполнять интегрирование еще одной задачи Коши (1.28), (1.29).

1.2 Частный вариант краевых условий

Зачастую достаточную трудность представляет решение алгебраических задач (1.13), (1.14) с прямоугольными матрицами, то есть определение векторов φ_k^0, φ_k^s $(s = 1, \ldots, r)$. От точности решения таких задач, естественно, зависит точность решения краевой задачи.

Наиболее просто решаются краевые задачи с так называемыми поточечно разделенными краевыми условиями, когда на концах отрезка $[a, b]$ заданы значения функций. В этом случае задача (1.1)–(1.3) записывается в виде

$$\frac{dy_i(x)}{dx} = \sum_{k=1}^{n} A_{ik}(x) y_k(x) + f_i(x), \quad i = 1, \ldots, n, \quad a < x < b, \tag{1.30}$$

$$y_i(a) = b_i, \quad i = 1, \ldots, n - r, \tag{1.31}$$

$$y_{i_k}(b) = c_{i_k}, \quad k = 1 \ldots, r, \quad 1 \leqslant i_k \leqslant n, \quad i_1 < i_2 \cdots < i_r. \tag{1.32}$$

Замечание (об индексах). На левом конце отрезка при $x = a$ задано $(n - r)$ краевых условий. Переменные y_i всегда можно перенумеровать так, чтобы индекс i принимал значения $i = 1, \ldots, n-r$. На правом конце отрезка при $x = b$ задано r краевых условий. В частности, номера переменных, для которых заданы краевые условия на правом конце отрезка, могут совпадать с номерами переменных, заданных на левом конце

отрезка. Этим объяснятся сравнительная громоздкость записи краевых условий (1.32) при $x = b$.

Конечно, краевая задача (1.30)–(1.32) совпадает с (1.1)–(1.3). Пусть, например, $n = 4$, $r = 3$. Тогда при $x = a$ задано $n - r$ краевых условий, то есть одно краевое условие (после перенумерации индексов — условие для y_1). При $x = b$ задано r краевых условий, например, для переменных y_1, y_3, y_4. В этом случае $i_1 = 1$, $i_2 = 3$, $i_3 = 4$. Матрицы $\boldsymbol{B}$ и $\boldsymbol{C}$ имеют вид

$$\boldsymbol{B} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}, \quad \boldsymbol{C} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.33}$$

Для краевых условий (1.31), (1.32) векторы $\boldsymbol{\varphi}^0$, $\boldsymbol{\varphi}^s$ $(s = 1, \ldots, r)$ записываются в форме

$$\boldsymbol{\varphi}^0 = (b_1, \ldots, b_{n-r}, 0, \ldots, 0)^T, \tag{1.34}$$

$$\boldsymbol{\varphi}^s = (0, \ldots, 0, 1_{n-r+s}, 0, \ldots, 0)^T, \quad s = 1, \ldots, r. \tag{1.35}$$

Здесь символ T означает транспонирование (то есть $\boldsymbol{\varphi}^0$, $\boldsymbol{\varphi}^s$ это вектор-столбцы); 1_{n-r+s} означает, что единица является $(n-r+s)$ компонентной, тогда как все остальные компоненты равны нулю; количество компонент каждого вектора равно n.

В этом случае схема решения имеет следующий вид.

1. Решаем задачу Коши для неоднородных уравнений (1.1)

$$\frac{dY_i^0(x)}{dx} = \sum_{k=1}^{n} A_{ik}(x) Y_k^0(x) + f_i(x), \quad i = 1, \ldots, n, \quad a < x < b, \tag{1.36}$$

с неоднородными начальными условиями

$$Y_i^0(a) = b_i, \quad i = 1, \ldots, n - r. \tag{1.37}$$

$$Y_i^0(a) = 0, \quad i = n - r + 1, \ldots, n. \tag{1.38}$$

2. Решаем серию ($s = 1, \ldots, r$) задач Коши для однородных уравнений (1.1)

$$\frac{dY_i^s(x)}{dx} = \sum_{k=1}^{n} A_{ik}(x) Y_k^s(x), \quad i = 1, \ldots, n, \quad a < x < b, \tag{1.39}$$

с неоднородными начальными условиями

$$Y_{n-r+s}^s(a) = 1, \tag{1.40}$$

$$Y_i^s(a) = 0, \quad i = 1, \ldots, n-r+s-1, n-r+s+1, \ldots, n. \tag{1.41}$$

3. Общее решение задачи Коши (1.1), (1.2) записывается в виде (см. формулу (1.22))

$$y_k(x) = Y_k^0(x) + \sum_{s=1}^{r} \theta_s Y_k^s(x), \quad k = 1, \ldots, n. \tag{1.42}$$

Подставляя (1.42) в краевые условия (1.32), имеем

$$Y_{i_k}^0(b) + \sum_{s=1}^{r} \theta_s Y_{i_k}^s(b) = c_{i_k}, \tag{1.43}$$

$$k = 1 \ldots, r, \quad 1 \leqslant i_k \leqslant n, \quad i_1 < i_2 \cdots < i_r.$$

Вычисляем матрицу M_{ks} и вектор m_k (сравни с формулами (1.25), (1.26))

$$M_{ks} = Y_{i_k}^s(b), \quad k, s = 1, \ldots, r. \tag{1.44}$$

$$m_k = c_{i_k} - Y_{i_k}^0(b), \quad k = 1, \ldots, r. \tag{1.45}$$

Напомним, что в формулах (1.44), (1.45) требуются значения решений задач Коши (1.36)–(1.41) лишь при $x = b$, то есть величины $Y_{i_k}^0(b)$, $Y_{i_k}^s(b)$ ($s = 1, \ldots, r$).

Решая систему алгебраических уравнений

$$\sum_{k=1}^{n} M_{ks} \theta_s = m_k, \quad k = 1, \ldots, r, \tag{1.46}$$

определяем θ_s.

4. Решение краевой задачи (1.1)–(1.3), после того как найдены параметры θ_s, определяется решением задачи Коши для уравнений

$$\frac{dy_i(x)}{dx} = \sum_{k=1}^{n} A_{ik}(x) y_k(x) + f_i(x), \quad i = 1, \ldots, n, \quad a < x < b, \tag{1.47}$$

с начальными условиями

$$y_i(a) = b_i, \quad i = 1, \ldots, n - r. \tag{1.48}$$

$$y_{n-r+s}(a) = \theta_s, \quad s = 1, \ldots, r. \tag{1.49}$$

1.3 Пример

Приведем пример некоторой конкретной краевой задачи с поточечно разделенными краевыми условиями.

$$\frac{dy_1}{dx} = -y_1 - y_2 + \sin(x), \quad a < x < b, \tag{1.50}$$

$$\frac{dy_2}{dx} = -y_2 + y_3, \quad a < x < b, \tag{1.51}$$

$$\frac{dy_3}{dx} = -y_2, \quad a < x < b, \tag{1.52}$$

$$y_1(a) = c_1, \tag{1.53}$$

$$y_2(b) = c_2, \quad y_3(b) = c_3, \tag{1.54}$$

где c_1, c_2, c_3 — заданные константы.

Для указанной задачи далее приведен численный алгоритм решения и его программная реализация. Для удобства программирования произведено некоторое изменение в обозначениях по сравнению с формулами п. 1.2. Эти изменения касаются констант, задающих краевые условия. В формулах (1.31), (1.32) константы при $x = a$ обозначены символами b_i, а константы при $x = b$ — символами c_i. В приведенном примере все константы в краевых условиях (1.53), (1.54) обозначены символами c_i. Кроме этого изменена нумерация параметров θ_s, которые заменены на p_s, и верхние индексы функций $Y_k^s(x)$ замененены римскими цифрами.

1.4 Алгоритм решения

Для решения задачи (1.50)–(1.54) используем следующий алгоритм.

1. Решаем неоднородные уравнения (1.50)–(1.52)

$$\frac{dy_1}{dx} = -y_1 - y_2 + \sin(x), \quad \frac{dy_2}{dx} = -y_2 + y_3, \quad \frac{dy_3}{dx} = -y_2, \tag{1.55}$$

с начальными условиями

$$y_1(a) = c_1, \quad y_2(a) = 0, \quad y_3(a) = 0. \tag{1.56}$$

Введем обозначение для решение задачи Коши (1.55), (1.56)

$$y_k(x) = Y_k^0(x), \quad k = 1, 2, 3. \tag{1.57}$$

2. Решаем однородные уравнения (1.50)–(1.52)

$$\frac{dy_1}{dx} = -y_1 - y_2, \quad \frac{dy_2}{dx} = -y_2 + y_3, \quad \frac{dy_3}{dx} = -y_2 \tag{1.58}$$

с начальными условиями

$$y_1(a) = 0, \quad y_2(a) = 1, \quad y_3(a) = 0. \tag{1.59}$$

Введем обозначение для решение задачи Коши (1.58), (1.59)

$$y_k(x) = Y_k^I(x), \quad k = 1, 2, 3. \tag{1.60}$$

3. Решаем однородные уравнения (1.50)–(1.52)

$$\frac{dy_1}{dx} = -y_1 - y_2, \quad \frac{dy_2}{dx} = -y_2 + y_3, \quad \frac{dy_3}{dx} = -y_2 \tag{1.61}$$

с начальными условиями

$$y_1(a) = 0, \quad y_2(a) = 0, \quad y_3(a) = 1. \tag{1.62}$$

Введем обозначение для решение задачи Коши (1.61), (1.62)

$$y_k(x) = Y_k^{II}(x), \quad k = 1, 2, 3. \tag{1.63}$$

4. Общее решение краевой задачи (1.50)–(1.54) при помощи решений задач Коши Y_k^0, Y_k^I, Y_k^{II} записывается в виде линейной комбинации решений

$$y_k(x) = Y_k^0(x) + p_2 Y_k^I(x) + p_3 Y_k^{II}(x), \quad k = 1, 2, 3, \tag{1.64}$$

где p_2, p_3 — некоторые неизвестные параметры.

5. Для определения параметров p_2, p_3 используем краевые условия (1.54), то есть условия при $x = b$. Подставляя, получим систему линейных уравнений относительно неизвестных p_2, p_3

$$y_2(b) = Y_2^0(b) + p_2 Y_2^I(b) + p_3 Y_2^{II}(b) = c_2, \tag{1.65}$$

$$y_3(b) = Y_3^0(b) + p_2 Y_3^I(b) + p_3 Y_3^{II}(b) = c_3, \tag{1.66}$$

которую, вводя обозначения,

$$b_1 = c_2 - Y_2^0(b), \quad a_{11} = Y_2^I(b), \quad a_{12} = Y_2^{II}(b), \tag{1.67}$$

$$b_2 = c_3 - Y_3^0(b), \quad a_{21} = Y_3^I(b), \quad a_{22} = Y_3^{II}(b), \tag{1.68}$$

удобно записать в виде

$$a_{11} p_2 + a_{12} p_3 = b_1, \tag{1.69}$$

$$a_{21} p_2 + a_{22} p_3 = b_2. \tag{1.70}$$

Решая (1.69), (1.70), получим значения параметров p_2, p_3.

Замечание. Нет необходимости иметь информацию о решениях задач Коши $Y_k^0(x)$, $Y_k^I(x)$, $Y_k^{II}(x)$ на всем интервале $a \leqslant x \leqslant b$. Достаточно знать решения лишь в точке $x = b$, то есть $Y_k^0(b)$, $Y_k^I(b)$, $Y_k^{II}(b)$.

6. Окончательное решение исходной краевой задачи (1.50)–(1.54), получим, решая задачу Коши

$$\frac{dy_1}{dx} = -y_1 - y_2 + \sin(x), \quad \frac{dy_2}{dx} = -y_2 + y_3, \quad \frac{dy_3}{dx} = -y_2, \tag{1.71}$$

с начальными условиями

$$y_1(a) = c_1, \quad y_2(a) = p_2, \quad y_3(a) = p_3. \tag{1.72}$$

2 Реализация программы

Предполагается, что на компьютере установлены одна из версий интерпретатора Python 3.5 и инсталлированы библиотеки tkinter, numpy, scipy, matplotlib. В случае отсутствия библиотек их можно инсталлировать (см, например, п. 4.2)

Основная цель программы — решение краевой задачи (1.50)–(1.54) с заданными параметрами c_1, c_2, c_3, a, b

$$\frac{dy_1}{dx} = -y_1 - y_2 + \sin(x), \quad a < x < b, \tag{2.1}$$

$$\frac{dy_2}{dx} = -y_2 + y_3, \quad a < x < b, \tag{2.2}$$

$$\frac{dy_3}{dx} = -y_2, \quad a < x < b, \tag{2.3}$$

$$y_1(a) = c_1, \tag{2.4}$$

$$y_2(b) = c_2, \quad y_3(b) = c_3. \tag{2.5}$$

Этапы достижения цели следующие:

a) создание интерфейса для ввода параметров c_1, c_2, c_3, a, b;

b) решение задачи (2.1)–(2.5) (алгоритм описан в п. 1.2);

c) визуализация решения.

Конечно, п. a) является, в некотором смысле, избыточным «излишеством», так как решается вполне конкретная задача. Создание развитого интерфейса, скорее, требуется для общих задач, например, вида (1.1)–(1.3), когда уравнения и краевые условия записаны в самом общем виде и достаточно лишь указать параметры задачи. Тем не менее, для того чтобы показать пример создания интерфейса далее реализуется и п. a).

В приводимом ниже листинге кодов программы использована сквозная нумерация. Это означает, что собрав все фрагменты кода в порядке нумерации, читатель получит полностью рабочую программу (разумеется, удалив нумерацию строк). Следует заметить, что если фрагмент оканчивается пустыми строками, то такие строки не приводятся и не нумеруются.

2.1 Импорт модулей

Для начала работы следует импортировать из модулей требуемые функции, что осуществляется в следующем фрагменте программы

```
1 # Метод пристрелки
2 import numpy as np, matplotlib.pyplot as plt, \
3     matplotlib.font_manager as fm,\
4     os
5 from scipy.integrate import odeint
6 from scipy import linalg
7 import matplotlib.patches as mpatches
8 import matplotlib.lines as mlines
```

Обратим внимание, что нет необходимости импортировать модуль полностью. Достаточно указать требуемые функции, которые из него извлекаются. Например, модуль **numpy** импортируется полностью и в дальнейшем ему присваиваться сокращенное обозначение **np**

```
import numpy as np
```

Напротив, из модуля **matplolib** импортируются лишь необходимые фрагменты, например,

```
import matplotlib.patches as mpatches
```

Назначение тех или иных модулей будет указано в дальнейшем при использовании функций из соответствующих модулей.

Заметим также, что импорт модулей можно осуществлять непосредственно перед вызовом из них требуемых функций, то есть необязательно в начале программы.

2.2 Интерфейс на основе tkinter

Для создания интерфейса, позволяющего вводить параметры c_1, c_2, c_3, a, b, используется модуль **tkinter**. Помимо указанных параметров, задается параметр N — количество точек, в которых на отрезке $[a, b]$ можно получить решение дифференциальных уравнений (подробнее см. строки 170–181 кода программы).

Результатом работы указанного ниже фрагмента программы является виджет для ввода параметров (см. рис. 1).

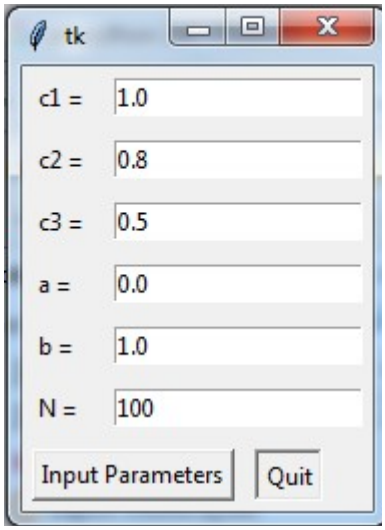

Рис. 1: Виджет ввода параметров

```
10
11 from tkinter import *
12 fields = ('c1', 'c2', 'c3', 'a', 'b', 'N')
13
14 def insert_parameters(entries):
15     global c1, c2, c3, a, b, nn
16
17     c1 = (float(entries['c1'].get()))
18     print("c1 = ", c1)
19
```

```
20      c2 = (float(entries['c2'].get()))
21      print("c2 = ", c2)
22
23      c3 = (float(entries['c3'].get()))
24      print("c3 = ", c3)
25
26      a = (float(entries['a'].get()))
27      print("a = ", a)
28
29      b = (float(entries['b'].get()))
30      print("b = ", b)
31
32      nn = (int(entries['N'].get()))
33      print("N = ", nn)
34
35      c1 = ("%8.2f" % c1).strip()
36      entries['c1'].delete(0,END)
37      entries['c1'].insert(0, c1)
38      print("c1 = %f" % float(c1))
39
40      c2 = ("%8.2f" % c2).strip()
41      entries['c2'].delete(0, END)
42      entries['c2'].insert(0, c2)
43      print("c2 = %f" % float(c2))
44
45      c3 = ("%8.2f" % c3).strip()
46      entries['c3'].delete(0, END)
47      entries['c3'].insert(0, c3)
48      print("c3 = %f" % float(c3))
49
```

```
50      a = ("%8.2f" % a).strip()
51      entries['a'].delete(0, END)
52      entries['a'].insert(0, a)
53      print("a = %f" % float(a))
54
55      b = ("%8.2f" % b).strip()
56      entries['b'].delete(0, END)
57      entries['b'].insert(0, b)
58      print("b = %f" % float(b))
59
60      #nn = nn.strip()
61      entries['N'].delete(0, END)
62      entries['N'].insert(0, nn)
63      print("N = ", int(nn))
64
65  def makeform(root, fields):
66      global c1, c2, c3, a, b, nn
67      entries = {}
68      for field in fields:
69          row = Frame(root)
70          lab = Label(row, width=5,
71                      text=field+" = ",
72                      anchor='w')
73          ent = Entry(row)
74          #ent.insert(0,"1")
75          row.pack(side=TOP, fill=X, padx=5, pady=5)
76          lab.pack(side=LEFT)
77          ent.pack(side=RIGHT, expand=YES, fill=X)
78          entries[field] = ent
79
```

```
80      #entries['c1'].delete(0, END)
81      entries['c1'].insert(0, c1)
82      #entries['c2'].delete(0, END)
83      entries['c2'].insert(0, c2)
84      #entries['c3'].delete(0, END)
85      entries['c3'].insert(0, c3)
86      #entries['a'].delete(0, END)
87      entries['a'].insert(0, a)
88      #entries['b'].delete(0, END)
89      entries['b'].insert(0, b)
90      # entries['N'].delete(0, END)
91      entries['N'].insert(0, nn)
92
93      return entries
94
95  if __name__ == '__main__':
96      global c1, c2, c3, a, b, nn
97      c1 = 1.0
98      c2 = 0.8
99      c3 = 0.5
100     a =0.0
101     b = 1.0
102     nn =100
103
104     root = Tk()
105     ents = makeform(root, fields)
106
107     button1 = Button(root, text='Input Parameters',
108                      command=(lambda e=ents:
109                               insert_parameters(e)))
```

```
110     button1.pack(side=LEFT, padx=5, pady=5)
111     button2 = Button(root, text='Quit',
112                      command=root.destroy)
113     #button2 = Button(root, text='Quit',
114     # command=root.quit)
115     button2.pack(side=LEFT, padx=5, pady=5)
116
117     root.mainloop()
118
119 print('c1 =', c1)
120 print('c2 =', c2)
121 print('c3 =', c3)
122 print('a  =', a)
123 print('b  =', b)
124 print('N  =', nn)
```

Дадим лишь краткое описание отдельных частей фрагмента.

Создаются поля ввода виджета

```
12 fields = ('c1', 'c2', 'c3', 'a', 'b', 'N')
```

Создаются две вспомогательные функции: функция

```
14 def insert_parameters(entries):
```

при помощи, которой осуществляется ввод параметров в заданные поля, и функция, создающая основную форму виджета,

```
65 def makeform(root, fields):
```

Для чтение строки из поля виджета, например, для величины c_1, служит фрагмент

```
17     c1 = (float(entries['c1'].get()))
18     print("c1 = ", c1)
```

Собственно говоря, строка 18 требуется лишь для контроля правильности ввода информации (при отладке программы) и может быть опущена.

Ввод остальных параметров реализуется аналогичным образом (см. строки 20–33).

Фрагмент (и аналогичные для других величин)

```
35    c1 = ("%8.2f" % c1).strip()
36    entries['c1'].delete(0,END)
37    entries['c1'].insert(0, c1)
38    print("c1 = %f" % float(c1))
```

осуществляет преобразование строки в формат чисел с плавающей точкой `%8.2f` (более точно, это всего лишь шаблон вывода в поле виджета, так как величина `c1`, по прежнему, является строковой величиной), поле вывода очищается, и в это поле заносится введенная величина. Строка 38 служит, вновь, только для контроля и не является обязательной.

Заметим, что `c1`, `c2`, `c3`, `a`, `c`, `N` являются строковыми переменными и необходимы дополнительные преобразования, если требуется получить численные величины.

Обратим также внимание на то, что функция `insert_parameters` не заканчивается, как обычно, возвратом каких либо-значений, то есть после строки 63 отсутствует `return`.

Функция `makeform` создает основную форму виджета и осуществляет ввод чисел в поля формы (см., например, строка 81).

Вызов виджета на дисплей осуществляется при помощи конструкции

```
95 if __name__ == '__main__':
```

Инициализируются значения переменных для первоначального вызова в поля формы (для того чтобы при вызове не появилась форма с пустыми полями)

```
97    c1 = 1.0
98    c2 = 0.8
99    c3 = 0.5
100   a =0.0
101   b = 1.0
102   nn =100
```

Осуществляется создание формы

```
104   ents = makeform(root, fields)
```

Создаются кнопки управления (см. рис.1):

для вызова функции `insert_parameters`

```
107   button1 = Button(root, text='Input Parameters',
108                    command=(lambda e=ents:
109                             insert_parameters(e)))
110   button1.pack(side=LEFT, padx=5, pady=5)
```

и для выхода из виджета

```
111   button2 = Button(root, text='Quit',
112                    command=root.destroy)
113   #button2 = Button(root, text='Quit',
114   # command=root.quit)
115   button2.pack(side=LEFT, padx=5, pady=5)
```

В данном случае, нажатие кнопки `Quit` повлечет прекращение работы виждета и его исчезновение с экрана. При необходимости сохранения виждета на экране дисплея следует заменить строки 111, 112 строками 113, 114. Заметим, что программа продолжит работу при любом способе выхода. Однако, если в дальнейшем программа должна будет осуществлять вывод в графические окна, то такой вывод не будет происходить до тех пор, пока не будет уничтожено окно виджета.

Строки

```
119 print('c1 =', c1)
120 print('c2 =', c2)
121 print('c3 =', c3)
122 print('a  =', a)
123 print('b  =', b)
124 print('N  =', nn)
```

служат для контроля работы программы и могут быть опущены.

Обратим внимание на то, что после работы данного виджета величины `c1`, `c2`, `c3`, `a`, `b`, `N` имеют строковый тип и для использования их в качестве чисел требуются дополнительные преобразования.

Отметим также, что строки

```
15     global c1, c2, c3, a, b, nn
```

```
66     global c1, c2, c3, a, b, nn
```

```
96     global c1, c2, c3, a, b, nn
```

служат для связи между функциями при помощи глобальных переменных. Строго говоря, такой подход не совсем соответствует духу программирования на языке Python. По возможности следует избегать задания глобальных переменных, и связь между функциями следует осуществлять при помощи параметров функций.

Примечание. Естественно, помимо модуля `tkinter`, для разработки `GUI` (`Graphical User Interface`) имеются и иные, возможно более хорошие модули, в частности, `PyQt5`, а также `wxpython` для `Python 2.7` и его модификация `wxpython Phoenix` для `Python 3+`.

2.3 Численное решение задачи на основе numpay, scipy

В этом разделе описана основная часть программы — численное решение задачи (2.1)–(2.5) методом пристрелки. Параметры c_1, c_2, c_3 и границы отрезка $[a, b]$ считаются известными. В частности, их можно задавать при помощи виджета, описанного в п. 2.2. Однако, как уже говорилось, непосредственно для проведения численных расчетов виджет носит лишь вспомогательный характер — его роль заключается в определении параметров при помощи интерфейсной части программы. Иными словами, вся часть программы между строками 10–125 может быть без ущерба опущена и заменена прямым заданием параметров непосредственно в тексте программы.

Далее последовательно приведен текст программы начиная со строки 126 до строки 304, после которой начинается графическая визуализация расчетов.

Строки

```
126
127 """
128 Решение системы НЕОДНОРОДНЫХ
129 линейных уравнений методом пристрелки
130 Система (пример)
131 dy1/dx =  - y1 - y2 + theta*f1(x)
132 dy2/dx =  - y2 + y3 + theta*f2(x)
133 dy3/dx =  - y2      + theta*f3(x)
134
135 Параметр theta = 0 соответствует ОДНОРОДНОЙ системе
136 Параметр theta = 1 соответствует НЕОДНОРОДНОЙ системе
137
138 Краевые условия (отрезок [a,b])
139
140 y1(a) = c1
```

```
141 y2(b) = c2
142 y3(b) = c3
143
144 fk(x) -- некоторые известные функции от x
145 В частности, в примере f1(x) = sin(x)
146
147 """
```

это обычный комментарий, в котором описана решаемая задача (2.1)–(2.5). Параметр `theta` введен для удобства, чтобы не записывать отдельно однородные (`theta=0`) и неоднородные (`theta=1`) дифференциальные уравнения.

Фрагмент

```
149 # Параметры (пример)
150 c1 = float(c1)
151 c2 = float(c2)
152 c3 = float(c3)
153
154 # Границы отрезка
155
156 a = float(a)
157 b = float(b)
```

описывает задание параметров `c1`, `c2`, `c3`, `a`, `b`, полученных из виджета. Напомним, что использованный вариант виджета позволяет подлучить `c1`, `c2`, `c3`, `a`, `b` строкового типа. Для преобразования `c1`, `c2`, `c3`, `a`, `b` в тип `float` и служит указанный фрагмент.

В случае, когда виджет не будет использоваться, строки 149–157 следует заменить обычным заданием величин `c1`, `c2`, `c3`, `a`, `b`, например,

```
c1 = 1.0
c2 = 0.8
```

```
c3 = 0.5
a  = 0.0
b  = 1.0
```

Следующий фрагмент задает начальные данные для решения серии задач Коши (см. п. 1.4)

```
159 # Начальные условия
160
161 # Для НЕОДНОРОДНОЙ задачи
162 initial_state_0 = [a, c1, 0.0, 0.0]
163
164 # Для ОДНОРОДНОЙ задачи I
165 initial_state_I = [a, 0.0, 1.0, 0.0]
166
167 # Для ОДНОРОДНОЙ задачи II
168 initial_state_II = [a, 0.0, 0.0, 1.0]
```

Переменные `initial_state_0`, `initial_state_I`, `initial_state_II` в дальнейшем используются при вызове функции решения системы дифференциальных уравнений.

Формат задания начальных данных достаточно прозрачен

$$[a, y_1(a), y_2(a), y_3(a)]. \tag{2.6}$$

Начальные данные, определенные фрагментом 159–168, соответствуют начальными данными (1.56), (1.59), (1.62).

Начало ($x = a$) и конец ($x = b$) отрезка на котором решается задача, а также количество точек отрезка, в которых следует получать решение задачи, определяется фрагментом

```
170 # Задание границ отрезка
171 start_x = a
172 end_x = b
173
```

```
174 # Количество точек для выдачи решения на отрезке [a,b]
175 N = int(nn)
```

Строка 175 преобразует строковую величину `nn`, полученную из виджета, в целочисленную (`int`) переменную `N`. Если виджет не будет использоваться, то строку 178 следует заменить, например, на

```
N = 100
```

Следующий фрагмент использует модуль `numpy` (сокращенно `np` см. строку 2). Грубо говоря, в переменную `x_points` формируется набор `N` точек на отрезке `[start_x, end_x]`. В принципе, строки 179–181 могут быть опущены — они предназначены для указания свойств формируемого набора

```
177 # Волшебный набор для использования odeint
178 x_points = np.linspace(start_x, end_x, num=N,
179                         endpoint=True,
180                         retstep=False,
181                         dtype=None)
```

Конкретная решаемая система обыкновенных уравнений (2.1)–(2.3) задается при помощи функции `systemODE` (имя может быть любым)

```
183 # Вариант решаемой системы
184
185 def systemODE(current_state, x):
186     # переменные y1, y2, y3; x = y0
187     y0, y1, y2, y3 = current_state
188
189     # define the 3 ordinary differential equations
190     global theta
191     dy0_dx = 1
192     dy1_dx = -y1-y2 + theta*np.sin(y0)
193     dy2_dx = -y2+y3
```

```
194     dy3_dx = -y2
195
196     # return a list of the equations
197     #  that describe the system
198     return [dy0_dx, dy1_dx, dy2_dx, dy3_dx]
```

Формат записи системы дифференциальных уравнений достаточно очевиден. Однако, следует сделать оговорку относительно решаемой системы. Неавтономная система уравнений (2.1)–(2.3), то есть с правой частью зависящей от x, преобразуется в автономную систему

$$\begin{aligned} \frac{dy_0}{dx} &= 1, \\ \frac{dy_1}{dx} &= -y_1 - y_2 + \theta \sin(y_0), \\ \frac{dy_2}{dx} &= -y_2 + y_3, \\ \frac{dy_3}{dx} &= -y_2. \end{aligned} \tag{2.7}$$

Роль переменной x в правой части уравнений играет величина y_0, которая, как это следует из первого уравнения системы совпадает с x, то есть $y_0(x) \equiv x$. Кроме этого, вводится дополнительный глобальный параметр θ (`theta`), который, как уже говорилось, позволяет превращать неоднородную систему ($\theta = 1$) в однородную систему ($\theta = 0$).

Строки 191–194 представляют собой почти дословную запись уравнений (2.8). Строка 187 указывает набор переменных для системы.

Имена идентификаторов `dy0_dx`, `dy1_dx`, `dy2_dx`, `dy3_dx` могут быть выбраны произвольно (это просто обозначения для правых частей уравнений) и должны быть выходными параметрами функции (строка 198).

В случае, когда потребуется решать систему уравнений, отличную от (2.8), ее легко записать при помощи указанной функции.

Для решения системы обыкновенных дифференциальных уравнений можно использовать функцию `odeint` из модуля `scipy`. Подробное опи-

сание всех возможностей функции `odeint` можно найти, например, по адресу

`https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html`

Здесь ограничимся лишь простейшим вариантом использования функции `odeint`

```
odeint(systemODE, initial_state, x_points)
```

Здесь

`systemODE` — функция, задающая систему дифференциальных уравнений (см. строки 185–198),

`initial_state` — величина, задающая начальные условия, например, одна из строк 162, 165, 168,

`x_points` — величина, задающая разбиение отрезка $[a, b]$ (см. строку 178).

Возможно чуть более общее использование функции с указанием точности решаемых уравнений

```
odeint(systemODE, initial_state, x_points,
       rtol = 1e-10, atol =1e-10)
```

Здесь `rtol`, `atol` — относительная и абсолютная точность решения (по умолчанию `1.49012e-8`).

Результатом работы функции `odeint` является `array of array` (массив массивов). Способ извлечения массивов для отдельных переменных показан в следующем фрагменте, который реализует решение неоднородной задачи Коши (1.55), (1.56)

```
200 # Решение НЕОДНОРОДНОЙ системы -- theta = 1
201
202 theta = 1.0
203
204 y0y1y2y3 = odeint(systemODE,
205                   initial_state_0, x_points)
```

```
206 # extract the individual arrays of y0, y1, y2, y3
207 #  values from the array of arrays
208 y0 = y0y1y2y3[:, 0]
209 y1 = y0y1y2y3[:, 1]
210 y2 = y0y1y2y3[:, 2]
211 y3 = y0y1y2y3[:, 3]
```

Полученное численное решение $Y_k^0(x)$ (см. (1.57)) содержится в массивах y1, y2, y3.

Требуемые для дальнейшего величины $Y_2^0(b)$, $Y_3^0(b)$ (см. (1.67), (1.68)) извлекаются при помощи фрагмента

```
213 # Извлечение требуемых для решения задачи значений
214 # Y20 = Y2(b), Y30 = Y3(b)
215
216 Y20 = y2[N-1]
217 Y30 = y3[N-1]
```

Аналогичным образом решается однородная задача Коши (1.58), (1.59) с сохраняемым решением $Y_k^I(x)$ (см. (1.60)) и извлечением $Y_2^I(b)$, $Y_3^I(b)$ (см. (1.67), (1.68))

```
219 # Решение ОДНОРОДНОЙ системы -- theta = 0, задача I
220
221 theta = 0.0
222
223 y0y1y2y3 = odeint(systemODE,
224                   initial_state_I, x_points)
225 # extract the individual arrays of y0, y1, y2, y3
226 #  values from the array of arrays
227 y0 = y0y1y2y3[:, 0]
228 y1 = y0y1y2y3[:, 1]
229 y2 = y0y1y2y3[:, 2]
```

```
y3 = y0y1y2y3[:, 3]

# Извлечение требуемых для решения задачи значений
# Y2I = Y2(b), Y3I = Y3(b)

Y2I = y2[N - 1]
Y3I = y3[N - 1]
```

Для решения однородной задача Коши (1.61), (1.62) с сохраняемым решением $Y_k^{II}(x)$ (см. (1.63)) и извлечением $Y_2^{II}(b)$, $Y_3^{II}(b)$ (см. (1.67), (1.68)) используется фрагмент

```
# Решение ОДНОРОДНОЙ системы -- theta = 0, задача II

theta = 0.0

y0y1y2y3 = odeint(systemODE,
                  initial_state_II, x_points)
# extract the individual arrays of y0, y1, y2, y3
#  values from the array of arrays
y0 = y0y1y2y3[:, 0]
y1 = y0y1y2y3[:, 1]
y2 = y0y1y2y3[:, 2]
y3 = y0y1y2y3[:, 3]

# Извлечение требуемых для решения задачи значений
# Y2I = Y2(b), Y3I = Y3(b)

Y2II = y2[N - 1]
Y3II = y3[N - 1]
```

Для формирование системы алгебраических уравнений (см. (1.67)–(1.70)) используем фрагмент

```
257 # Формирование системы линейных
258 # АЛГЕБРАИЧЕСКИХ уравнений для определния p2, p3
259 b1 = c2 - Y20
260 b2 = c3 - Y30
261 A = np.array([[Y2I, Y2II], [Y3I, Y3II]])
262 bb = np.array([[b1], [b2]])
263
264 print(A)
265 print(bb)
```

Решение системы (1.69), (1.70) осуществляется при помощи функции `linalg.solve` модуля `numpy`

```
267 # Решение системы
268
269 p2, p3 = np.linalg.solve(A, bb)  # fast
270 print(p2)
271 print(p3)
```

Следующий фрагмент, как и строки 270, 271, в принципе, не требуются и сохранен лишь для контроля работы программы

```
273 # Проверка
274
275 print(A.dot(np.linalg.solve(A, bb)) - bb)
```

Приведем для справки значения p2, p3, которые получаются при использовании параметров, встроенных в виджет по умолчанию (см. строки 97–101, а также матрицу и правые части системы (1.69), (1.70)

```
A = [[ 0.12619295  0.5335072 ]
     [-0.5335072   0.65970014]]
b = [[ 0.8]
     [ 0.5]]
p1 = [ 0.70948927]
p2 = [ 1.33169235]
```

Окончательное решение краевой задачи реализовано в следующем фрагменте (решение задачи Коши (1.71), (1.72))

```
277 # Окончательное решение краевой
278 # НЕОДНОРОДНОЙ задачи, theta = 1
279
280 theta = 1.0
281
282 initial_state = [a, c1, p2, p3]
283
284 y0y1y2y3 = odeint(systemODE, initial_state, x_points)
285 # extract the individual arrays
286 # y0, y1, y2, y3  values
287 # from the array of arrays
288 y0 = y0y1y2y3[:, 0]
289 y1 = y0y1y2y3[:, 1]
290 y2 = y0y1y2y3[:, 2]
291 y3 = y0y1y2y3[:, 3]
```

Фрагмент

```
293 # Проверка
294
295 print('y0[0]=', y0[0])
296 print('y1[0]=', y1[0])
297 print('y2[0]=', y2[0])
298 print('y3[0]=', y3[0])
299
300 print('y0[N-1]=', y0[N-1])
301 print('y1[N-1]=', y1[N-1])
302 print('y2[N-1]=', y2[N-1])
303 print('y3[N-1]=', y3[N-1])
```

требуется лишь для контроля полученного решения. В частности, проверяется выполнение краевых условий.

Для справки укажем полученные значения

```
y0[0]= 0.0
y1[0]= 1.0
y2[0]= 0.709489266113
y3[0]= 1.33169235464

y0[N-1]= 1.0
y1[N-1]= 0.173507804325
y2[N-1]= 0.799999991414
y3[N-1]= 0.500000009555
```

Результаты вычисления сохранены в массивах `y1`, `y2`, `y3`. Заметим, что массив `y0` содержит точки, в которых сохранены значения функций `y1`, `y2`, `y3`.

Реализованный алгоритм в некотором смысле удачен с точки зрения сохранения памяти. В частности, решения «промежуточных» задач Ко-

ши (строки 208–211, 227–230, 246–249) не сохраняются и окончательно решение, вновь, записывается в уже использованные массивы `y0`, `y1`, `y2`, `y3` (строки 288–291). Однако, с точки зрения скорости расчетов такой алгоритм не является оптимальным, так как для получения окончательного решения приходится решать дополнительную задачу Коши (см. строки 277–291).

Выполнение расчетов можно было бы ускорить, сохраняя результаты решений промежуточных задач Коши, то есть функции $Y_k^0(x)$, $Y_k^I(x)$, $Y_k^{II}(x)$ (см. (1.57), (1.60), (1.63)). В этом случае решение могло бы быть записано при помощи соотношения (1.64), разумеется после нахождения параметров p_2, p_3. При этом дополнительного решения задачи Коши (1.71), (1.72) не потребуется.

Еще одним «недостатком» использованного алгоритма является «раздельное» решение задач Коши для нахождения функций $Y_k^0(x)$, $Y_k^I(x)$, $Y_k^{II}(x)$. Имеется в виду, что каждая задача Коши решается независимо друг от друга и точности построенных решений, в принципе, могут отличаться друг от друга. Тактически более грамотно было бы объединить решение всех задач Коши в одну задачу. Для этого достаточно ввести новые обзначения

$$\begin{aligned} &u_1(x) = y_1^0(x), \quad u_2(x) = y_2^0(x), \quad u_3(x) = y_3^0(x), \\ &u_3(x) = y_1^I(x), \quad u_4(x) = y_2^I(x), \quad u_6(x) = y_3^I(x), \\ &u_7(x) = y_1^{II}(x), \quad u_8(x) = y_2^{II}(x), \quad u_9(x) = y_3^{II}(x). \end{aligned} \tag{2.8}$$

Задача Коши для новых переменных имеет вид

$$\begin{aligned} &\frac{du_1}{dx} = -u_1 - u_2 + \sin(x), \quad \frac{du_2}{dx} = -u_2 + u_3, \quad \frac{du_3}{dx} = -u_2, \\ &\frac{du_4}{dx} = -u_4 - u_5, \quad \frac{du_5}{dx} = -u_5 + u_6, \quad \frac{du_6}{dx} = -u_5, \\ &\frac{du_7}{dx} = -u_7 - u_8, \quad \frac{du_8}{dx} = -u_8 + u_9, \quad \frac{du_9}{dx} = -u_8, \end{aligned} \tag{2.9}$$

$$u_1(a) = c_1, \quad u_2(a) = 0, \quad u_3(a) = 0, \tag{2.10}$$
$$u_4(a) = 0, \quad u_5(a) = 1, \quad u_6(a) = 0,$$
$$u_7(a) = 0, \quad u_8(a) = 0, \quad u_9(a) = 1.$$

При решении такой задачи все величины $u_k(x)$ будут вычисляться с одинаковой точностью.

2.4 Визуализация решения на основе matplotlib

Модуль `matplotlib` дает широкие возможности для визуализации результатов расчета, сохранения изображений и т. п. Подробное описание модуля с множеством примеров имеется, например, по адресу

`http://matplotlib.org/`

Здесь ограничимся лишь кратким описанием фрагментов программы, позволяющих представить результаты расчетов в графической форме. На рис. 2 показан вывод результатов в одно графическое окно, которое разделено на два подокна.

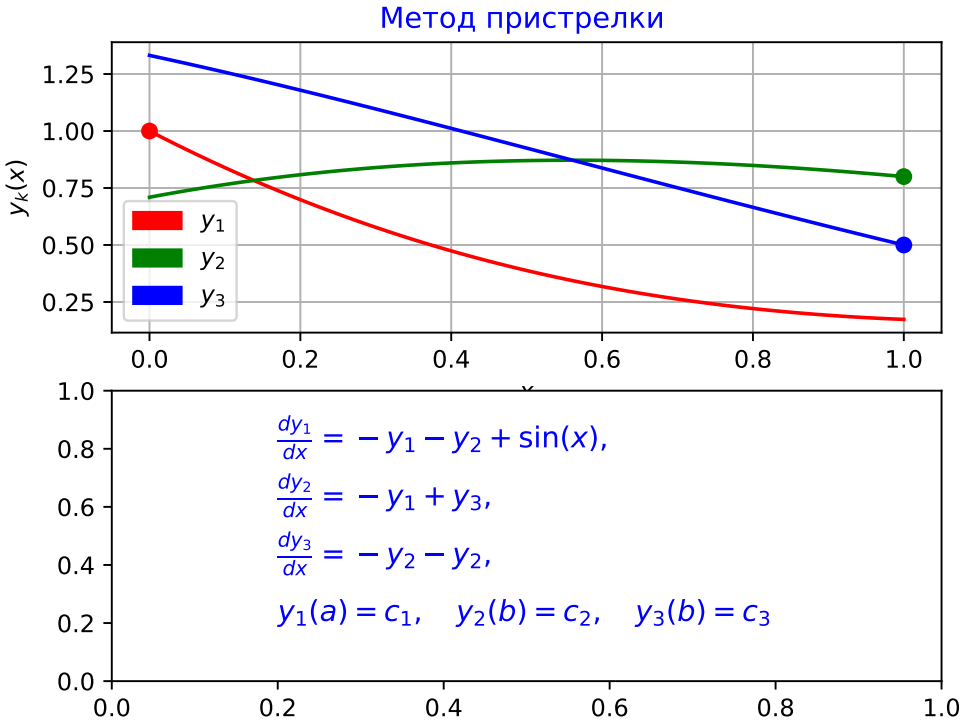

Рис. 2: Результаты расчетов

На рис. 3 показан результат вывода в одно окно анимированного изображения, имитирующего процесс пошагового решения задачи Коши.

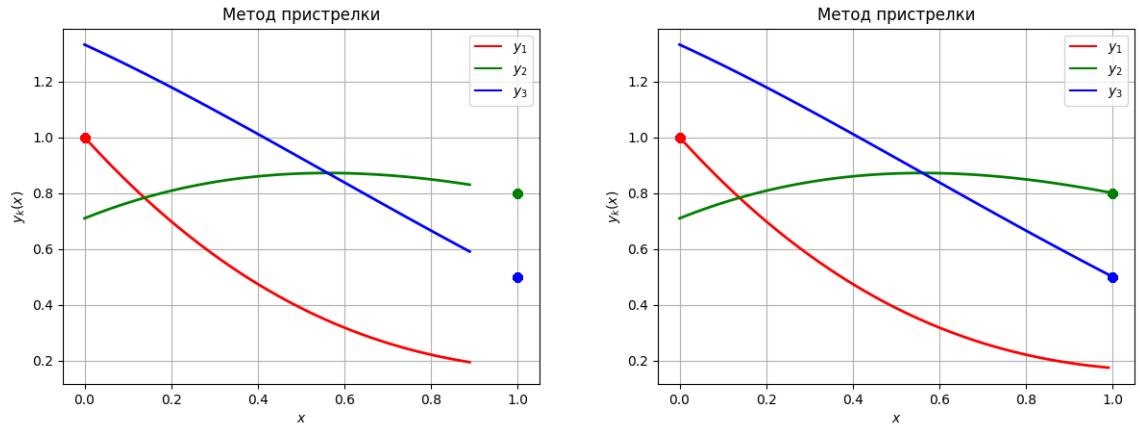

Рис. 3: Результаты расчетов. Анимация процесса решения краевой задачи

2.4.1 Визуализация статического изображения

Результаты расчетов, хранящиеся в массивах `y0`, `y1`, `y2`, `y3` (см. строки 288–291), сохраняем в массивы `xx`, `yy1`, `yy2`, `yy3`, которые будут использоваться при вызове графических функций модуля `matplotlib`,

```
305 # Визуализация расчетов
306
307 j = N
308 xx = y0[:j]
309 yy1 = y1[:j]
310 yy2 = y2[:j]
311 yy3 = y3[:j]
```

Напомним, что при решении задач Коши использовалось разбиение отрезка $[a, b]$ на набор точек (см. строки 178–181)

$$[x_0, x_1, \ldots, x_N], \tag{2.11}$$

$$x_k = a + k\frac{b-a}{N}, \quad k = 0, \ldots, N, \tag{2.12}$$

в которых сохранялись значения функций $y_1(x)$, $y_2(x)$, $y_3(x)$.

Иными словами массивы `y0`, `y1`, `y2`, `y3` имеют вид

$$\texttt{y0} = [x_0, x_1, \ldots, x_N], \tag{2.13}$$

$$\texttt{y1} = [y_1(x_0), y_1(x_1), \ldots, y_1(x_N)], \tag{2.14}$$

$$\texttt{y2} = [y_2(x_0), y_2(x_1), \ldots, y_2(x_N)], \tag{2.15}$$

$$\texttt{y3} = [y_3(x_0), y_3(x_1), \ldots, y_3(x_N)]. \tag{2.16}$$

Создание новых массивов `xx`, `yy1`, `yy2`, `yy3` позволяет при визуализации, если потребуется, строить графики на полном отрезке $[a, b]$, а также на его части, то есть отрезке $[x_0, x_j]$.

Строка

```
313 plt.subplot(2, 1, 1)
```

указывает на то, что изображение будет выводится в верхнее подокно графического окна (см. рис. 2).

Команда

`subplot(numrows,numcols,fignum)`

указывает на какое количество подокон будет разбито основное окно.

Параметры команды определяют положение подокон

`numrows` — количество строк (количество подокон по вертикали)

`numcols` — количество столбцов (количество подокон по горизонтали)

`fignum` — номер подокна, в которое будет происходить вывод (изменяется от `1` до `numrows*numcols`)

Следующие строки

```
315 plt.plot([a], [c1], 'ro')
316 plt.plot([b], [c2], 'go')
317 plt.plot([b], [c3], 'bo')
```

позволяют построить на графике точки с координатами, соответственно, (a, c_1), (b, c_2), (b, c_3) (см. рис. 2). Параметры `'ro'`, `'go'`, `'bo'` указывают, что построены будут именно точки (символ `o` — «кружок») красного (red), зеленого (green) и синего (blue) цвета.

Непосредственно построение графиков осуществляется при помощи фрагмента

```
319 plt.plot(xx, yy1, color='r') #Построение графика
320 plt.plot(xx, yy2, color='g') #Построение графика
321 plt.plot(xx, yy3, color='b') #Построение графика
```

Следующий фрагмент

```
323 plt.xlabel(r'$x$') #Метка по оси x в формате TeX
324 plt.ylabel(r'$y_k(x)$') #Метка по оси y в формате TeX
325
326 plt.title(r'Метод пристрелки ', color='blue')
327 plt.grid(True) #Сетка
```

указывает способ построения меток на осях, заголовка рисунка и сетки (см. рис. 2).

Вывод на рисунок подписей (легенды) в виде закрашенных прямоугольников с указанием переменных осуществляется при помощи фрагмента (см. рис. 2).

```
330 patch_y1 = mpatches.Patch(color='red',
331                           label='$y_1$')
332 patch_y2 = mpatches.Patch(color='green',
333                           label='$y_2$')
334 patch_y3 = mpatches.Patch(color='blue',
335                           label='$y_3$')
336
337 plt.legend(handles=[patch_y1, patch_y2, patch_y3])
```

При необходимости можно определять номер максимального (или минимального) элемента массива и его номер

```
339 # Определение номера максимального
340 # элемента массива и его значения
341 #nb = np.argmax(xx)
342 #print(nb)
```

```
343 #print(xx[nb])
```

Следующий фрагмент позволяет получить размеры графика, воспроизводимого в верхнем окне.

```
345 # Получение размеров графика
346 ymin, ymax = plt.ylim()
347 print(ymin, ymax)
348 xmin, xmax = plt.xlim()
349 print(xmin, xmax)
```

Необходимость в таких размерах возникает в случае, когда в каких-либо графических окнах, например при анимации, требуется восстановить прежние реальные размеры графика. Обратим внимание на то, что если в новых окнах график будет выводится не в подокно, а в полное графическое окно, то на осях будут воспроизводится истинные размеры (такие как были в подокне).

Следующий фрагмент (из серии «излишества») демонстрирует способ создания нижнего подокна, в котором воспроизводится информация о задаче

```
350
351 plt.subplot(2, 1, 2)
352
353 font = {'family': 'serif',
354         'color':  'blue',
355         'weight': 'normal',
356         'size': 12,
357         }
358
359 plt.text(0.2, 0.8,
360          r'$\frac{dy_1}{dx}= - y_1 - y_2 + \sin(x),$',
361          fontdict=font)
```

```
362 plt.text(0.2, 0.6,
363          r'$\frac{dy_2}{dx}= - y_1 + y_3,$',
364          fontdict=font)
365 plt.text(0.2, 0.4,
366          r'$\frac{dy_3}{dx}= - y_2 - y_2,$',
367          fontdict=font)
368 plt.text(0.2, 0.2,
369          r'$y_1(a)=c_1, '
370          r'\quad y_2(b)=c_2, \quad y_3(b)=c_3.$',
371          fontdict=font)
372
373 # Задание отсупа для меток
374 plt.subplots_adjust(left=0.15)
```

Наконец, строка

```
376 plt.show() #Показать график
```

указывает, что сформированные прежними фрагментами объекты должны быть показаны в графическом окне (на дисплее).

2.4.2 Визуализация анимированного изображения

Модуль `matplotlib` предоставляет возможность создавать анимированные изображения. Применительно к рассматриваемой краевой задаче можно, например, проследить процесс решения задачи Коши по мере движения траекторий $y_k(x)$ от левого конца отрезка $x = a$ до правого конца $x = b$.

Для этого потребуется функция `animation`, которую следует импортировать из модуля `matplotlib`

```
378 from matplotlib import animation
```

Фрагмент

```
380 fig = plt.figure()
381
382 ax = plt.axes(xlim=(xmin, xmax), ylim=(ymin, ymax))
383 line1, = ax.plot([], [], 'r', lw=2)
384 line2, = ax.plot([], [], 'g', lw=2)
385 line3, = ax.plot([], [], 'b', lw=2)
```

описывает создание графического окна `fig`, указывает, что графики будут изображаться в окне с размерами (`xmin`, `xmax`) и (`ymin`, `ymax`) (эти величины были ранее получены в строках 346–349), и создает объекты `line1,`, `line2,`, `line3,`, в которые в дальнейшем будут заноситься данные расчетов для анимации. Параметр `'lw'` задает толщину линии (line width).

Обратим внимание на конструкцию (и аналогичные)

```
383 line1, = ax.plot([], [], 'r', lw=2)
```

Функция `plot()` возвращает кортеж объектов. Запятая после `line1,` означает, что переменной `line1` присваивается именно объект, а не кортеж из одного объекта. В частности, это означает, что можно обращаться к свойствам объекта с помощью вызова его методов.

Создается функция

```
386 # animation function.  This is called sequentially
387 def animate(i):
```

указывающая все объекты, предназначенные для графического окна, в котором будет воспроизводится анимация — метки на осях, сетка, точки, легенда (ср. со строками 315–317, 323–327, 330–337)

```
388     plt.xlabel(r'$x$') # Метка по оси x формат TeX
389     plt.ylabel(r'$y_k(x)$') # Метка по оси y
390     plt.title(r'Метод пристрелки') # Заголовок в TeX
391
```

```
392     plt.grid(True)  # Сетка
393     plt.plot([a], [c1], 'ro')
394     plt.plot([b], [c2], 'go')
395     plt.plot([b], [c3], 'bo')
396
397
398     red_line = mlines.Line2D([], [],
399                              color='red',
400                              label='$y_1$')
401     green_line = mlines.Line2D([], [],
402                                color='green',
403                                label='$y_2$')
404     blue_line = mlines.Line2D([], [],
405                               color='blue',
406                               label='$y_3$')
407     plt.legend(handles=
408                [red_line, green_line, blue_line])
```

Фрагмент

```
410     x = xx[:i]
411     y = yy1[:i]
412     line1.set_data(x, y)
413
414     x = xx[:i]
415     y = yy2[:i]
416     line2.set_data(x, y)
417
418     x = xx[:i]
419     y = yy3[:i]
420     line3.set_data(x, y)
```

описывает способ вывода линий в графическое окно. Понятно, что `x` и `y` содержат не на полную линию, а лишь ее часть (отрезок $[x_0, x_i]$). Именно таким способом в графическом окне будут построены линии в зависимости от параметра функции `i` (см. строку 387) линии. При `i=1` будет построена линия на отрезке $[x_0, x_1]$, при `i=2` будет построена линия на отрезке $[x_0, x_2]$, при `i=3` будет построена линия на отрезке $[x_0, x_3]$ и т. д. Очевидно, что при изменении параметра `i` будут достигнут эффект анимации изображения.

Возвращаемыми величинами функции будут объекты `line1, line2, line3,`

```
422     return line1, line2, line3,
```

Анимация осуществляется при помощи функции `animation.FuncAnimation`

```
424 # call the animator.
425 # blit=True означает, что перерисовывается
426 # лишь изменяющаяся часть рисунка.
427 anim = animation.FuncAnimation(fig, animate,
428                                frames=N,
429                                interval=120,
430                                blit=True)
```

Назначение параметров достаточно очевидно.

`fig` — указывает графическое окно, в котором будет осуществляться анимация (см. строку 380),

`animate` — это имя функции, указывающей на то, что и как будет изображаться в графическом окне (см. строки 386–422)

`frames` — задает максимальное число фреймов для анимации. Фактически, функция `animation.FuncAnimation` — это некий «итератор», который последовательно изменяет параметр `i` функции `animate` от `i=0` до `i=frames`

`interval` — задает интервал времени, через который будут изменяться фреймы (следует подбирать, исходя от тактовой частоты процессора компьютера).

`blit` — параметр, определяющий способ анимации. При `blit=True` все объекты, которые были изображены на `i`-ом фрейме, на следующем `(i+1)`-ом фрейме **перерисовываться не будут**. Иными словами, при изменении `i` в графическом окне перерисовывается только изменяющаяся часть рисунка. Это чрезвычайно важная особенность функции `animation.FuncAnimation`, позволяющая существенно экономить время, затрачиваемое программой на вывод в графическое окно.

Фрагмент

```
# To save the animation,
# use the command:
#anim.save('shoot02.mp4')
```

позволяет записывать анимацию в файл с указанным именем (естественно, следует убрать знаки комментария `#`).

Обратим внимание, что приведенный фрагмент будет работать лишь в случае, когда на компьютере установлен перекодировщик ffmpeg (перекодировщик, способ установки и связь перекодировщика с `Python` легко найти в интернете).

Приведенный ниже фрагмент позволяет осуществлять задержку анимации путем нажатия какой-либо клавиши или щелчком мыши в графическом окне. Указаны разные способы реализации такой возможности. Для наших целей нет нужды разбираться в способах реализации задержки. При необходимости, достаточно копировать приведенный фрагмент в соответствующую программу.

```
# Пауза/Продолжение анимации
#  при нажатии Space (Пробел)

anim_running = True
```

```
440
441 '''
442 def keypress(event):
443     global anim_running
444     if event.key == ' ': # SPACE: pause and resume
445         if anim and anim.event_source:
446             anim.event_source.stop()
447             if anim_running
448             else anim.event_source.start()
449             anim_running ^= True
450
451 fig.canvas.mpl_connect('key_press_event', keypress)
452
453 '''
454
455 def onClick(event):
456     global anim_running
457     if anim_running:
458         anim.event_source.stop()
459         anim_running = False
460     else:
461         anim.event_source.start()
462         anim_running = True
463
464 # Для нажатия клавишы
465 # fig.canvas.mpl_connect('key_press_event', onClick)
466
467 # Для нажатия мыши
468 fig.canvas.mpl_connect('button_press_event', onClick)
```

Наконец, строка

```
470 plt.show() #Показать график
```

позволяет вызвать процесс рисования (в данном случае, анимацию изображения).

Примечание. Строки 1–470 представляют полный код программы и при желании могут быть просто скопированы и выполнены при помощи интерпретатора Python. Напомним, что строки с пропущенной нумерацией являются пустыми строками. Например, в коде не указана пустая строка 469. Настоятельно рекомендуется выполнить приведенный код прежде, чем разбираться в его описании.

2.4.3 Вывод изображения в различные графические окна

Приведем код короткой программы, демонстрирующий способ создания двух (и более) различных графических окон. Программа, ввиду простоты, не нуждается в пояснениях. Укажем лишь, что строки 12 и 16 означают, что при выполнении рисования (см. строка 20) будут созданы два отдельных графических окна.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  a = 0.0
5  b = 2.0*np.pi
6
7  # Количество точек на отрезке [a,b]
8  N = 100
9
10 x_points = np.linspace(a, b, num=N)
11
12 plt.figure(1)
13 plt.plot(x_points, np.sin(x_points))
```

```
14 plt.grid(True) # Сетка
15
16 plt.figure(2)
17 plt.plot(x_points, np.cos(x_points))
18 plt.grid(True) # Сетка
19
20 plt.show()
21 # print(x_points)
```

Для того чтобы в программе метода пристрелки сделать одновременный вывод в два независимых графических окна, достаточно произвести следующие изменения.

1. Добавить в строке 312 `plt.figure(1)`
2. Закомментировать строку 376
3. Заменить строку 380 на `fig = plt.figure(2)`

3 Оформление скриптов при помощи LaTeX

В системе компьютерной верстки LaTeX имеется пакет `listing`, который позволяет оформлять скрипты Python, заключая их в рамки, делая нумерацию строк, обеспечивая выделение цветом ключевых слов. Собственно говоря, данный текст был оформлен именно при помощи этого пакета. Подробное описание возможностей имеется в руководстве к пакету. Здесь ограничимся лишь указанием фрагмента преамбулы и одной из команд пакета, действие которой достаточно очевидно (строки 69–73).

```
1  \documentclass[a4paper,12pt]{article}
2
3  \usepackage[cp1251]{inputenc}
4  \usepackage[russian]{babel}
5
6  \usepackage[left=2.2cm, right=2.8cm, %
7              top=2cm, bottom=3.1cm,   %
8              includehead]{geometry}   %
9
10 \jot=3mm
11 \flushbottom
12
13 \usepackage{graphicx}
14 \usepackage{color}
15 \usepackage{float}
16
17
18 % Default fixed font does not support bold face
19 % for bold
20 \DeclareFixedFont{\ttb}{T1}{txtt}{bx}{n}{12}
21 % for normal
22 \DeclareFixedFont{\ttm}{T1}{txtt}{m}{n}{12}
```

```
23
24 % Custom colors
25 \usepackage{color}
26 \definecolor{deepblue}{rgb}{0,0,0.5}
27 \definecolor{deepred}{rgb}{0.6,0,0}
28 \definecolor{deepgreen}{rgb}{0,0.5,0}
29
30 \usepackage{listings}
31
32 % Python style for highlighting
33 \newcommand\pythonstyle{\lstset{
34 language=Python,
35 basicstyle=\ttm,
36 bold black keywords
37 commentstyle=\ttfamily\color{blue}, % white comments
38 moredelim=*[s][\ttfamily\color{deepgreen}]{\'}{\'},
39 moredelim=*[s][\ttfamily\color{blue}]{\'\'\'}{\'\'\'},
40 otherkeywords={self},             % Add keywords here
41 keywordstyle=\ttb\color{deepblue},
42 emph={MyClass,__init__},          % Highlighting
43 emphstyle=\ttb\color{deepred},    % Highlighting style
44 stringstyle=\color{deepgreen},    %
45 numbers=left,                     %
46 numberstyle=\small,               %
47 stepnumber=1,                     %
48 numbersep=5pt,
49 xleftmargin=15mm,                 %
50 xrightmargin=5mm,                 %
51 frame=tb,                         % Any extra options
52 showstringspaces=false,           %
```

```
framexleftmargin=0mm,                %
framexrightmargin=-5mm,              %
frame=shadowbox,                     %
rulesepcolor=\color{blue}
}}

% Python for external files
\newcommand\pythonexternal[2][]{{
\pythonstyle
\lstinputlisting[#1]{#2}}}



\begin{document}
% ...

\pythonexternal[numbers=left,
                firstnumber=126,
                firstline=126,
                lastline=148]{Name.py}
\pythonexternal[numbers=left, firstnumber=1]{name.py}

% ...

\end{document}
```

4 Задания

В этом разделе приведены варианты задний.

4.1 Задача с поточечно разделенными краевыми условиями

№ 4.1.1. Реализовать программу для решения краевой задачи методом пристрелки (слева направо)

$$\frac{dy_1}{dx} = -\sin(5x)y_1 + \cos(3x)y_2 + y_3 + \sin(x), \quad 0 < x < 1, \tag{4.1}$$

$$\frac{dy_2}{dx} = xy_1 - xy_2 + xy_3 + x,$$

$$\frac{dy_3}{dx} = \sin(x)y_1 + \sin(7x)y_2 + 1,$$

$$y_1(0) = 1, \quad y_2(1) = 1.94898, \quad y_3(1) = 2.70362.$$

Данные для контроля:

$$y_1(0) = 1, \quad y_2(0) = 0, \quad y_3(0) = 1.$$

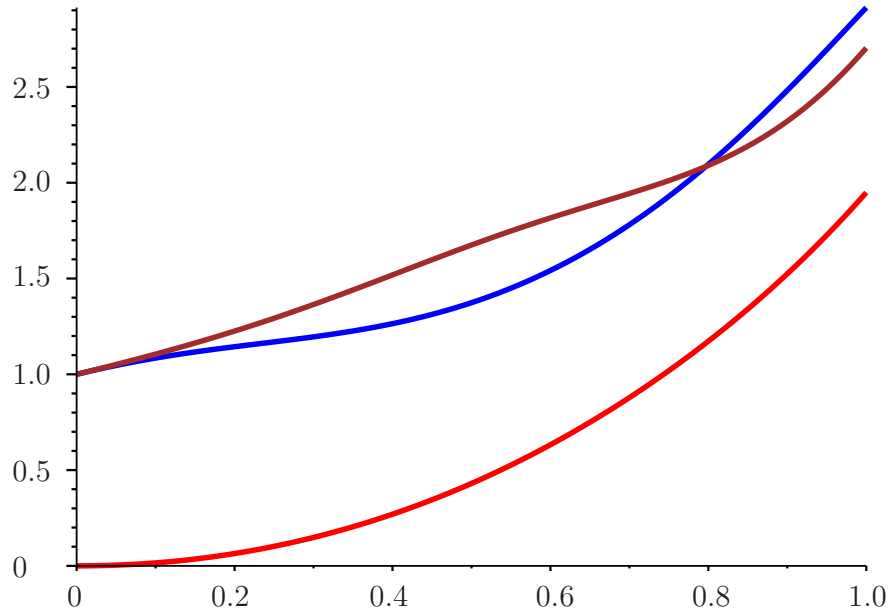

Рис. 4: Решение задания № 4.1.1. $y_1(x)$ — синий, $y_2(x)$ — красный, $y_3(x)$ — коричневый

№ 4.1.2. Реализовать программу для решения краевой задачи методом пристрелки (слева направо)

$$\frac{dy_1}{dx} = -e^{-x}\sin(5x)y_1 + xy_2 + y_3 + \cos(3x) + x, \quad 0 < x < 1, \tag{4.2}$$

$$\frac{dy_2}{dx} = y_1 + \cos(35x)y_2 + e^{-2x}y_3 + 1,$$

$$\frac{dy_3}{dx} = \cos(x)y_1 + y_2 + \sin(3x)y_3 + x^2,$$

$$y_1(0) = 1, \quad y_2(0) = 0.5, \quad y_3(1) = 4.53315.$$

Данные для контроля:

$$y_1(0) = 1, \quad y_2(0) = 0.5, \quad y_3(0) = 0.$$

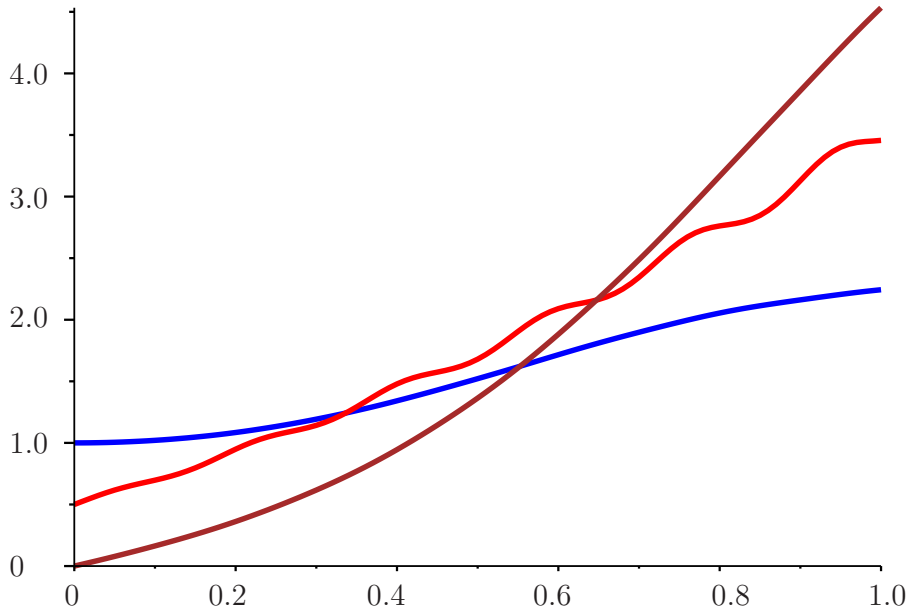

Рис. 5: Решение задания № 4.1.2. $y_1(x)$ — синий, $y_2(x)$ — красный, $y_3(x)$ — коричневый

№ 4.1.3. Реализовать программу для решения краевой задачи методом пристрелки (справа налево)

$$\frac{dy_1}{dx} = y_1 + 2y_2 - 3y_3 + 4, \quad 0 < x < 1, \tag{4.3}$$

$$\frac{dy_2}{dx} = -y_1 + y_2 + 1,$$

$$\frac{dy_3}{dx} = y_1 + 2y_2 + \sin(30x)y_3 + e^{-x},$$

$$y_1(0) = 1, \quad y_3(0) = 1.5, \quad y_3(1) = 6.56677.$$

Данные для контроля:

$$y_1(0) = 0, \quad y_2(0) = 1, \quad y_3(0) = 1.5.$$

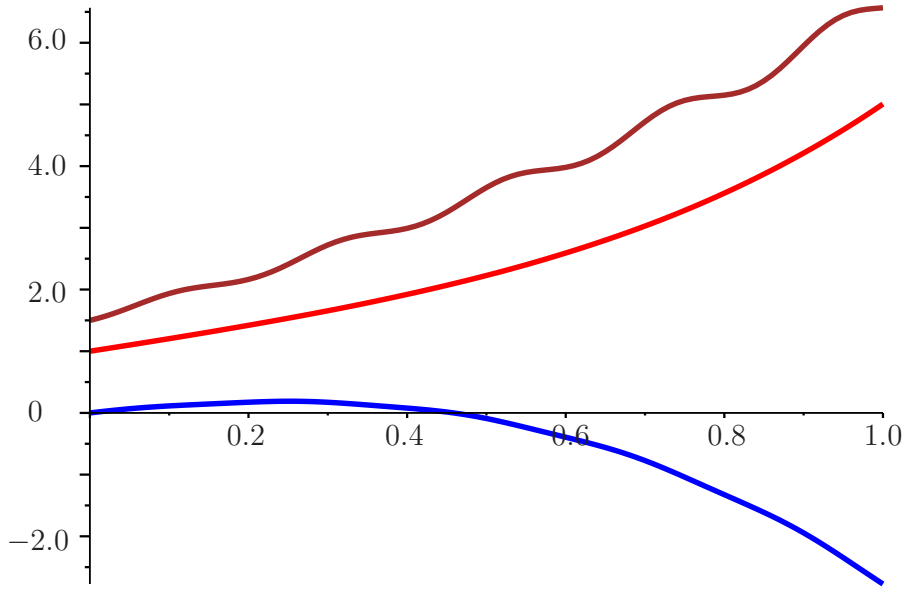

Рис. 6: Решение задания № 4.1.3. $y_1(x)$ — синий, $y_2(x)$ — красный, $y_3(x)$ — коричневый

4.2 Задача с краевыми условиями общего вида

№ 4.2.1. Реализовать программу для решения краевой задачи методом пристрелки (слева направо)

$$\frac{dy_1}{dx} = -y_1 + 2xy_2 + \sin(5x), \quad 0 < x < 1, \tag{4.4}$$

$$\frac{dy_2}{dx} = y_1 - xy_2 + y_3 + x,$$

$$\frac{dy_3}{dx} = y_1 - 2y_2,$$

$$y_1(0) - 2y_2(0) = 0, \quad y_2(0) - y_3(0) = 1,$$

$$y_1(1) + y_2(1) + y_3(1) = 3.87866.$$

Данные для контроля:

$$y_1(0) = 2, \quad y_2(0) = 1, \quad y_3(0) = 1.$$

$$y_1(1) = 2.44312, \quad y_2(1) = 2.51284, \quad y_3(1) = -1.07730.$$

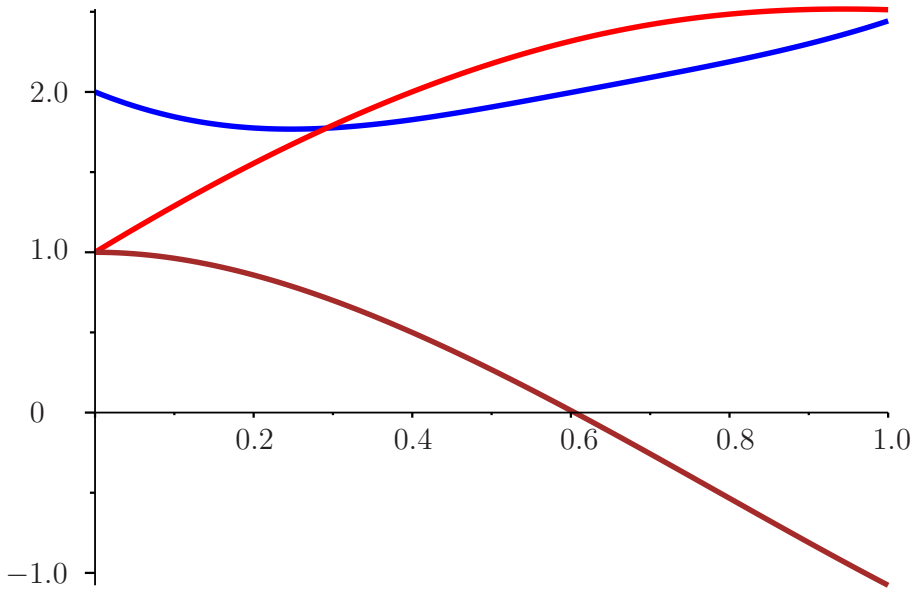

Рис. 7: Решение задания № 4.2.1. $y_1(x)$ — синий, $y_2(x)$ — красный, $y_3(x)$ — коричневый

Литература

1. **Бахвалов Н. С.** Численные методы. М.: Наука, 1973. 632 с.

2. **Калиткин Н. Н.** Численные методы. М.: Наука, 1978. 512 с.

Приложение.
Установка дополнительных библиотек

Отсутствующие библиотеки можно инсталлировать, вызывая командные строки, например, для numpy

```
pip install numpy
```

Замечание. Предполагается, что на компьютере имеется лишь одна версия Python и установлены соответствующие переменные окружения (Envirenment). В противном случае необходимо указывать путь к расположению `pip`.

При работе в оболочке PyCharm инсталляция дополнительных модулей может быть сделана следующим способом:

```
File -> Settings -> Project: -> Project Interpreter
```

В окне программы будет приведен список всех установленных модулей. Для инсталляции недостающего модуля следует нажать значок «+» (в правом верхнем углу) и в строке поиска указать имя отсутствующего модуля. Далее следует выбрать модуль, указать номер версии `Specify version`, и нажать кнопку Install Package. В случае удачной (или неудачной) установки появится соответствующее сообщение.

Список иллюстраций