

Отчёт по задаче №2.44

Орлов Даниил Ильич

December 2024

1 Постановка задачи

Требуется решить численно следующее дифференциальное уравнение:

$$\begin{cases} \frac{d^2x}{dt^2} + x^2 + \lambda t^2 = 1, \\ 0 < t < 1, \\ t = 0 : x = 0, \dot{x} = 1; \\ t = 1 : x = 0. \end{cases} \quad (1)$$

И найти минимальное λ , такое, при котором существует нетривиальное решение

2 Теоретическая часть

1. Будем решать данную задачу методом Рунге-Кутты с 5-м порядком погрешности. Для этого мы возьмём наше уравнение и начальные условия, чтобы получить задачу Коши.
2. Приведём наше уравнение к нужному виду, введя эти обозначения: $x = x_1$ и $\dot{x} = x_2$. Получим следующее:

$$\begin{cases} \dot{x}_2 = 1 - \lambda t^2 - x_1^2, \\ \dot{x}_1 = x_2, \\ 0 < t < 1, \\ t = 0 : x_1 = 0, x_2 = 1, \\ t = 1 : x_1 = 0; \end{cases}$$

3. Для нахождения λ мы запустим наш метод, начиная с λ_0 , проводя итерации по $\Delta\lambda$ с конечным значением в λ_{end} .
4. Чтобы получить нужные лямбда, мы построим график зависимости решения задачи в точке $x(1)$ от λ и нас будет интересовать точки λ , близкие к нулю

5. Вопрос минимальности, на мой взгляд не совсем уместен. Учитывая, что мы получаем малые значения, мы будем близки к точке $x(1) = 0$, но так как у нас лишь дискретное количество точек, то при уменьшении шага итерации $\Delta\lambda$ мы получим точки, более точные (в плане минимальности), чем получили до этого

3 Практическая часть

1. Ранее мы уже упоминали, что решать будем методом Рунге-Кутты 5-го порядка. Ниже написаны вложенные формулы Рунге, необходимые для решения задачи:

$$\begin{aligned}
 &\bullet \vec{k}_1 = hf(t, \vec{x}) \\
 &\bullet \vec{k}_2 = hf\left(t + \frac{h}{2}, \vec{x} + \frac{\vec{k}_1}{2}\right) \\
 &\bullet \vec{k}_3 = hf\left(t + \frac{h}{2}, \vec{x} + \frac{(\vec{k}_1 + \vec{k}_2)}{4}\right) \\
 &\bullet \vec{k}_4 = hf\left(t + h, \vec{x} - \vec{k}_2 + 2\vec{k}_3\right) \\
 &\bullet \vec{k}_5 = hf\left(t + \frac{2h}{3}, \vec{x} + \frac{(7\vec{k}_1 + 10\vec{k}_2 + \vec{k}_4)}{27}\right) \\
 &\bullet \vec{k}_6 = hf\left(t + \frac{h}{5}, \vec{x} + \frac{(28\vec{k}_1 - 125\vec{k}_2 + 546\vec{k}_3 + 54\vec{k}_4 - 378\vec{k}_5)}{625}\right)
 \end{aligned}$$

2. Ошибку будем вычислять следующим образом: $\vec{r} = -\frac{(42\vec{k}_1 + 224\vec{k}_3 + 21\vec{k}_4 - 162\vec{k}_5 - 125\vec{k}_6)}{336}$
3. Ошибка на шаге: $||\vec{r}||$
4. Вычисление значения функции на шаге: $\vec{x}(t+h) = \vec{x}(t) + \frac{(\vec{k}_1 + 4\vec{k}_3 + \vec{k}_4)}{6}$
5. Тестирование программы. Возьмем два примера из книги Э.Хайрер, С.Нёрсетт и Г.Ваннера "Решение обыкновенных дифференциальных уравнений Нежёсткие задачи ". А именно:

$$\begin{cases} \dot{x}_1 = x_2 x_3, & x_1(0) = 0, \\ \dot{x}_2 = -x_1 x_3, & x_2(0) = 1, \\ \dot{x}_3 = -0.51 x_1 x_2, & x_3(0) = 1, \\ 0 \leq t \leq 20. \end{cases}$$

Решая систему, можем получить следующий график:

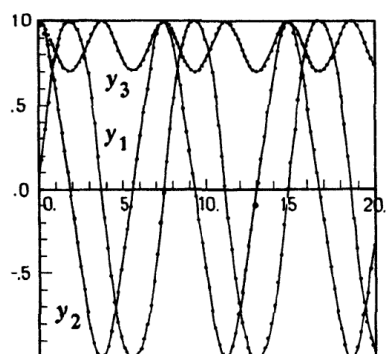


Рис. 1: Из книги.

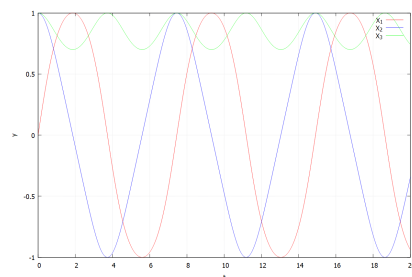


Рис. 2: Мой вариант при решении задачи.

Также рассмотрим ещё одну задачу из этой же книги:

$$\begin{cases} \dot{x}_1 = x_2, & x_1(0) = 2, \\ \dot{x}_2 = (1 - x_1^2)x_2 - x_1, & x_2(0) = 0, \\ 0 \leq t \leq 20. \end{cases}$$

Её решения приведём следующим графиком:

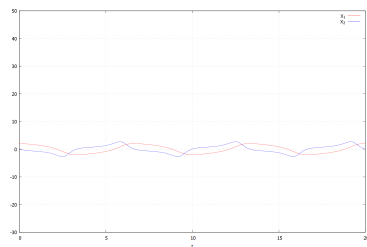


Рис. 3: Общий вид.

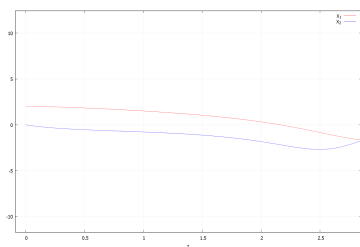


Рис. 4: Приближение.

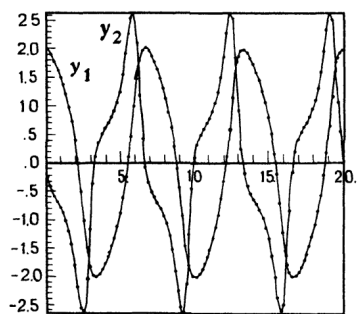


Рис. 5: Из книги.

Графики, которые я привёл, очень схожи, можно сделать вывод, что моя программа выдаёт корректный результат. Можно взять также другие готовые методы рунге, и просто сравнивать ошибку в результатах при вычислении решения уравнения, но я считаю, что результат будет схожим.

4 Три Просчёта:

Точность	Начальная точка	Конечная точка	Время работы(сек.)	Количество принятых шагов	Количество непринятых шагов	Глобальная ошибка
10^{-7}	(0, 1)	$(-6.78562 \cdot 10^{-7}, -3.89559)$	0.001	26	1	$1.85605 \cdot 10^{-6}$
10^{-9}	(0, 1)	$(-7.05262 \cdot 10^{-7}, -3.89559)$	0.001	26	1	$3.89963 \cdot 10^{-8}$
10^{-11}	(0, 1)	$(-7.06852 \cdot 10^{-7}, -3.89559)$	0.001	158	0	$9.24984 \cdot 10^{-10}$
10^{13}	(0, 1)	$(-7.0687 \cdot 10^{-7}, -3.89559)$	0.001	399	1	$2.33577 \cdot 10^{-11}$
$2 \cdot 10^{-16}$	(0, 1)	$(-7.06877 \cdot 10^{-7}, -3.89559)$	0.001	1284	2	$2.66553 \cdot 10^{-12}$

Легко видеть, что по трём просчётам сохраняется отношение погрешностей и Глобальных ошибок. Сами Ошибки мы считывали следующим образом:

1. Линеаризуем систему и выпишем матрицу Якоби.

$$A = \begin{pmatrix} 0 & 1 \\ -2 \cdot x_1 & 0 \end{pmatrix}$$

2. Высчитываем матрицу B , которая равна произведению матрицы A , на транспонированную A^T .

$$A = A \times A^T = \begin{pmatrix} 0 & 1 \\ -2 \cdot x_1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & -2 \cdot x_1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 4 \cdot x_1^2 \end{pmatrix}$$

3. Далее рассматриваем собственные значения данной матрицы. Они будут равны: $\lambda_1 = 1$ и $\lambda_2 = 4 \cdot x_1^2$ соответственно.
4. Берём максимальное $\lambda_{max,i} = \max \lambda_1, \lambda_2$ на каждом шаге.
5. Общая формула для вычисления глобальной ошибки на шаге: $errglob(t_i) = errloc(t_{i+1}) + errglob(t_i) \cdot \exp(\lambda_{max,i} \cdot h_{i+1})$

5 Результаты:

Программа работает довольно эффективно. Она гибкая, чтобы метод вычислял решение функции разной размерности, нужно лишь задать саму функцию, и размерность. Также учитывая, что я в решении использовал язык C++, а не чистый C(хотя разница в моей программе лишь в форме вывода), программа считает не много медленнее, чем та же программа на C. Причиной этому стал вывод программы на консоль, или запись файла, так как использовалась стандартная библиотека STL(в ней тот же `std::cout` - замедляет программу из-за того, что это объект ООП, а сам вывод - это

перегруженный метод $<<$, который тоже замедляет). Но несмотря на все, программа весьма "шустро" решает задачу.

Сначала запускаем программу для $\lambda = -100$ с шагом 10^{-3} , получаем следующие графики:

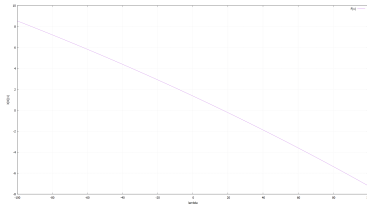


Рис. 6: Общий вид.

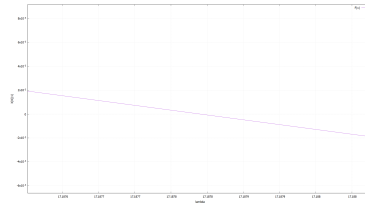


Рис. 7: Приближённый вариант.

Если посмотреть на графики, то одно из ближайших значений (итерация с шагом 10^{-3}) это $\lambda = 17.187$. Рассмотрим теперь итерирование с шагом 10^{-4} , получаем следующий график:

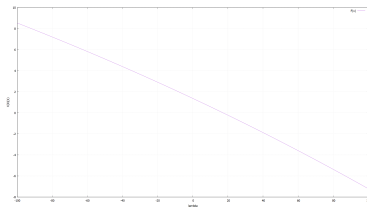


Рис. 8: Общий вид.

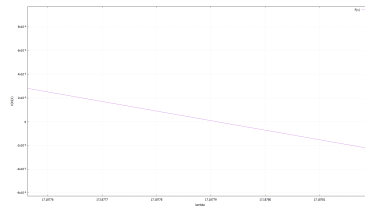


Рис. 9: Приближённый вариант.

Уже в этом случае, мы получаем, что минимальное значение, которое нам подходит, оно является $\lambda = 17.1878$

6 Оценки:

Теперь поступим следующим образом: мы выяснили, что наша точка точно находится в промежутке от $\lambda = 17$ до $\lambda = 18$, в нём точно корень. Тогда запустим так называемый "бинарный поиск" корня, и будем после деления пополам выбирать ту часть отрезка, в котором лежит корень. Осталось понять, когда остановится? Идея такая:

1. Для начала надо определить главный член погрешности, определим так:

$$\begin{aligned}
x(1, \lambda, \text{tol} = 10^{-7}) - x(1, \lambda) &= C \cdot 10^{-7q}, \\
x(1, \lambda, \text{tol} = 10^{-9}) - x(1, \lambda) &= C \cdot 10^{-9q}, \\
x(1, \lambda, \text{tol} = 10^{-7}) - x(1, \lambda) - x_0 10^{-7} &= C \cdot 10^{-7q}, \quad q = \frac{4}{5}, \\
\text{где } x(1, \lambda, \text{tol}) &- \text{значение функции } x[0](1) \text{ с погрешностью tol,} \\
x(1, \lambda) &- \text{искомое значение, } C \cdot 10^{-7q} - \text{соответствующий член погрешности.}
\end{aligned}$$

вычтем одно, из другого, можем с лёгкостью найти член погрешности.

2. Потом мы определим малую окрестность искомого λ , такое, чтобы уменьшаемый нами отрезок находился в нём полностью.

$$\begin{aligned}
\Delta\lambda &= \frac{C \cdot \text{tol}^q}{\frac{dx}{d\lambda}}, \\
\frac{dx}{d\lambda} &\approx \frac{x(1, \lambda + 10^{-4}) - x(1, \lambda - 10^{-4})}{2 \cdot 10^{-4}}
\end{aligned}$$

3. И последнее, смотрим, попал ли наш отрезок в границу $\Delta\lambda$: если попал, то берём в качестве ответа λ = середина отрезка, иначе продолжаем поиск корня.

Вернёмся к нашей задаче, запутив поиск, описанный выше для отрезка с корнем, а именно отрезок $[17, 18]$, получаем следующее: $\lambda = 17.187791$ для 4 знаков после запятой, $\lambda = 17.18779124$ для восьми, и $\lambda = 17.1877912439$ для 10 знаков соответственно.

В заключении также хочу сказать, что на точность, скорость работы программы влияет(существенно влияет) погрешность, которую мы принимаем на шаге, в моей работе ошибку я брал очень жёсткую, а именно $\text{tol} = 2 \cdot 10^{-16}$