Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Численные методы»
Вариант №1

Студент: А. О. Дубинин
Преподаватель: И. Э. Иванов
Группа: М8О-306Б

Дата:
Оценка:
Подпись:

Москва, 2020

Лабораторная работа №4

Вариант 1

1

Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки h. С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

№	Задача Коши	Точное решение
1	$y'' + y - sin3x = 0,$ $y(0) = 1,$ $y'(0) = 1,$ $x \in [0, 1], h = 0.1$	$y = cosx + \frac{11}{8} sinx - \frac{sin3x}{8}$

2

Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

№	Задача Коши	Точное решение
1	$xy'' + 2y' - xy = 0,$ $y'(1) = 0,$ $1.5y(2) + y'(2) = e^2,$	$y(x) = \frac{e^x}{x}$

1 Решение

1 Метод Эйлера, Рунге-Кутты и Адамса 4-го порядка

Рассматривается задача Коши для одного дифференциального уравнения первого порядка, разрешенного относительно производной

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

Требуется найти решение на отрезке [a,b], здесь $x_0 = a$. Введем разностную сетку на отрезке [a,b]:

$$\Omega^{(k)} = x_k = x_0 + hk, h = |b - a|/N, h = 0, 1, ..., N.$$

Точки x_k называются узлами разностной сетки, расстояние h между узлами – шагом разностной сетки, а совокупность заданных в узлах сетки значений какой-либо величины называется сеточной функцией y(h) = $y_k, k = 0, 1, ..., N$. Приближенное решение задачи Коши будем искать численно в виде сеточной функции $y^{(h)}$. Для оценки погрешности приближенного численного решения $y^{(h)}$ будем рассматривать это решение как элемент (N + 1) – мерного линейного векторного пространства с какой-либо нормой. В качестве погрешности решения принимается норма элемента этого пространства $\delta^{(h)} = y^{(h)} - [y]^{(h)}, [y]^{(h)}-$ точное решение задачи в узлах расчетной сетки. Таким образом $\epsilon_h = ||\delta^{(h)}||$.

Пусть необходимо решить задачу Коши для ОДУ второго порядка:

$$y'' = f(x, y, y'),$$

$$y(x_0) = y_0,$$
$$y'(x_0) = y_{01},$$

Путем введения замены $z = y'$ приведем к системе:

$$y' = z,$$

$$z' = f(x, y, z),$$
$$y(x_0) = y_0,$$
$$z(x_0) = y_{01},$$

Метод Эйлера

Метод Эйлера играет важную роль в теории численных методов решения ОДУ, хотя и не часто используется в практических расчетах из-за невысокой точности. Вывод расчетных соотношений для этого метода может быть произведен несколькими способами: с помощью геометрической интерпретации, с использованием разложения в ряд Тейлора, конечно разностным методом (с помощью разностной аппроксимации производной), квадратурным способом (использованием эквивалентного интегрального уравнения).

Рассмотрим вывод соотношений метода Эйлера геометрическим способом. Решение в узле x_0 известно из начальных условий рассмотрим процедуру получения решения в узле x_1

График функции $y^{(h)}$, которая является решением задачи Коши, представляет собой гладкую кривую, проходящую через точку (x_0, y_0) согласно условию $y(x_0) = y_0$, и имеет в этой точке касательную. Тангенс угла наклона касательной к оси Ox равен значению производной от решения в точке x_0 и равен значению правой части дифференциального уравнения в точке (x_0, y_0) согласно выражению $y'(x_0) = f(x_0, y_0)$. В случае небольшого шага разностной сетки h график функции и график касательной не успевают сильно разойтись друг от друга и можно в качестве значения решения в узле x_1 принять значение касательной y_1, вместо значения неизвестного точного решения y_1. При этом допускается погрешность $|y_1 - y_1|$ геометрически представленная отрезком CD на рис.4.1. Из прямоугольного треугольника ABC находим CB=BA tg(CAB) или $\Delta y = hy'(x_0)$. Учитывая, что $\Delta = y_1 - y_0$ и заменяя производную $y'(x_0)$ на правую часть дифференциального уравнения , получаем соотношение $y_1 = y_0 + hf(x_0, y_0)$. Считая теперь точку (x_1, y_1) начальной и повторяя все предыдущие рассуждения, получим значение y_2 в узле x_2.

Переход к произвольным индексам дает формулу метода Эйлера:

$$y_{k+1} = y_k + hf(x_k, y_k)$$

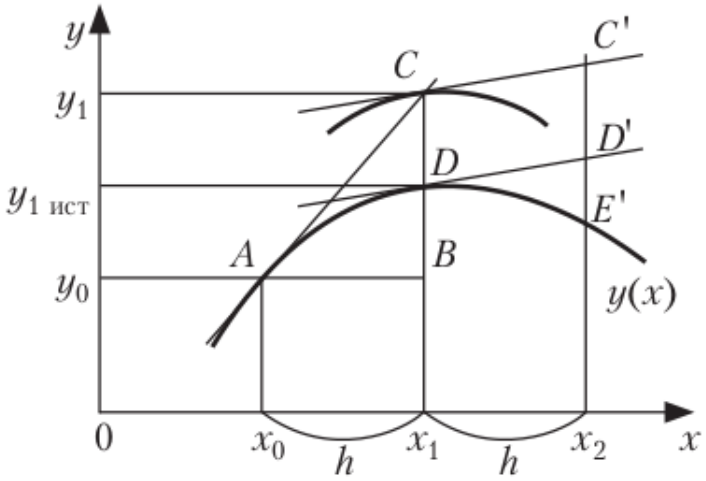

Рис. 4.1

Пример

Реализовать методы Эйлера в виде программы, задавая в качестве входных данных шаг сетки h. С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

№	Задача Коши	Точное решение
1	$y'' + y - sin3x = 0,$ $y(0) = 1,$ $y'(0) = 1,$ $x \in [0,1], h = 0.1$	$y = cosx + \frac{11}{8} sinx - \frac{sin3x}{8}$

Решение.

Итак, исходя из начальной точки $x_0 = 0, y_0 = 1, z_0 = y'_0 = 1$ рассчитаем значение z_1, y_1 в узле $x_1 = 0.1$ по формулам:

$$z_1 = z_0 + hf(x_0, y_0, y'_0) = 1 + 0.1 * (0.0 - 1) = 0.9$$

$$y_1 = y_0 + hz_1 = 1.09$$

Аналогично получим решение в следующем узле $x_2 = 0.2$:

$$z2 = z1 + hf(x1, y1, y'1) = 0.9 + 0.1 * (0.2955202066613396 - 1.09) = 0.820552020666134$$

$$y2 = y1 + hz = 1.1720552020666135$$

Продолжим вычисления и, введя обозначения $\Delta y_k = hz$ и $\epsilon_k = |y_{src}(x_k) - y_k|$, где $y_{src}(x_k)$ – точное решение в узловых точках, получаемые результаты занесем в таблицу

x	y	Δy_k	y_{src}	ϵ_k	Romberg's method error
0	1	-0.1	1.0	0.0	0.0
0.1	1.09	-0.0794479793	1.09533509	0.005335087334746	0.0903350873347
0.2	1.1720552	-0.0607412728	1.1826566	0.010601396443457	0.1692021099143
0.3	1.24803628	-0.0464709367	1.26376091	0.015724632735001	0.2397639782966
0.4	1.31937026	-0.0387331171	1.34000633	0.020636070977157	0.3043251166349
0.5	1.38683093	-0.0389335940	1.4121058	0.025274876803588	0.3639314154264
0.6	1.45039824	-0.0476550606	1.47998806	0.029589824645564	0.4181064569088
0.7	1.50920004	-0.0645990674	1.54274034	0.033540295166676	0.4647500739992
0.8	1.56154194	-0.0886078757	1.59863844	0.037096498356469	0.5002104698976
0.9	1.60502305	-0.1177643167	1.64526198	0.040238935979036	0.5195252544050
1.0	1.63672773	–	1.67968491	0.042957183057446	0.5168112676677

Метод Рунге – Кутты четвертого порядка

Семейство явных методов Рунге-Кутты p-го порядка записывается в виде совокупности формул:

$$y_{k+1} = y_k + \Delta x_k$$

$$\Delta y_k = \sum_{i=1}^{p} c_i K_i^k$$

$$K_i^k = hf(x_k + a_i h, y_k + h \sum_{j=1}^{i-1} b_{ij} K_j^k)$$

$$i = 2, 3...p$$

Параметры a_i, b_{ij}, c_i подбираются так, чтобы значение y_{k+1}, рассчитанное по соотношению совпадало со значением разложения в точке x_{k+1} точного решения в ряд Тейлора с погрешностью $O(h^{p+1})$

Метод Рунге-Кутты четвертого порядка точности

$p = 4, a_1 = 0, a_2 = \frac{1}{2}, a_3 = \frac{1}{2}, a_4 = 1, b_{21} = \frac{1}{2}, b_{31} = 0, b_{32} = \frac{1}{2}, b_{41} = 0, b_{42} = 0, b_{43} = \frac{1}{2}, c_1 = \frac{1}{6}, c_2 = \frac{1}{3}, c_3 = \frac{1}{3}, c_3 = \frac{1}{6}$

является одним из самых широко используемых методов для решения Задачи Коши:

$$y_{k+1} = y_k + \Delta y_k$$
$$\Delta y_k = \frac{1}{6}(K_1^k + 2K_2^k + 2K_3^k + K_4^k)$$
$$K_1^k = hf(x_k, y_k)$$
$$K_2^k = hf(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_1^k)$$
$$K_3^k = hf(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_2^k)$$
$$K_4^k = hf(x_k + h, y_k + K_3^k)$$

Пример

Реализовать методы Рунге-Кутты 4-го порядка в виде программы, задавая в качестве входных данных шаг сетки h. С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

№	Задача Коши	Точное решение
1	$y'' + y - sin3x = 0,$ $y(0) = 1,$ $y'(0) = 1,$ $x \in [0, 1], h = 0.1$	$y = cosx + \frac{11}{8}sinx - \frac{sin3x}{8}$

Решение.

Введем новую переменную $z = y'$ решение исходной начальной задачи для дифференциального уравнения второго порядка сводится к решению системы двух дифференциальных уравнений первого порядка.

$$y' = z$$
$$z' = sin3x - y$$
$$y(0) = 1$$
$$z(0) = 1$$

Вычислим значения вспомогательных величин

$$K_1^0 = hf(x_0, y_0, z_0) = hz_0 = 0.1$$
$$L_1^0 = hg(x_0, y_0, z_0) = h\frac{2x_0z_0}{x_0^2 + 1} = -0.1$$

$$K_2^0 = hf(x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}K_1^0, z_0 + \frac{1}{2}L_1^0) = 0.095$$

$$L_2^0 = hg(x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}K_1^0, z_0 + \frac{1}{2}L_1^0) = -0.09$$

$$K_3^0 = hf(x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}K_2^0, z_0 + \frac{1}{2}L_2^0) = 0.095$$

$$L_3^0 = hg(x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}K_2^0, z_0 + \frac{1}{2}L_2^0) = -0.089$$

$$K_4^0 = hf(x_0 + h, y_0 + K_3^0, z_0 + L_3^0) = 0.091$$

$$L_4^0 = hg(x_0 + h, y_0 + K_3^0, z_0 + L_3^0) = -0.079$$

Найдем приращения функций на первом интервале

$$\Delta y_0 = \frac{1}{6}(K_1^0 + 2K_2^0 + 2K_3^0 + K_4^0) = 0.095$$

$$\Delta z_0 = \frac{1}{6}(L_1^0 + 2L_2^0 + 2L_3^0 + L_4^0) = -0.090$$

и значения функций в первом узле

$$y_1 = y_0 + \Delta y_0 = 1.095$$

$$z_1 = z_0 + \Delta z_0 = 0.910$$

Аналогично получим решения в остальных узлах, результаты вычислений занесем в таблицу.

x	y	z	Δy_k	Δz_k	y_{src}	ϵ_k	Romberg's method error
0	1	1	0.09534	-0.08995	1.0	0.0	0.0
0.1	1.09534	0.91005	0.08732	-0.07062	1.09534	1e-06	0.04965
0.2	1.18266	0.83942	0.0811	-0.05446	1.18266	1e-06	0.09314
0.3	1.26376	0.78496	0.07625	-0.04381	1.26376	1e-06	0.13214
0.4	1.34001	0.74116	0.0721	-0.04043	1.34001	2e-06	0.16784
0.5	1.41211	0.70073	0.06788	-0.04533	1.41211	2e-06	0.20077
0.6	1.47999	0.6554	0.06275	-0.05864	1.47999	2e-06	0.23064
0.7	1.54274	0.59676	0.0559	-0.07962	1.54274	2e-06	0.25633
0.8	1.59864	0.51714	0.04662	-0.10672	1.59864	2e-06	0.27587
0.9	1.64526	0.41042	0.03442	-0.13772	1.64526	1e-06	0.28662
1.0	1.67969	0.27269	0	0	1.67968	1e-06	0.28542

Метод Адамса четвертого порядка

Многошаговые методы решения задачи Коши характеризуются тем, что решение в текущем узле зависит от данных не в одном предыдущем узле, как это имеет место в одношаговых методах, а от нескольких предыдущих узлах. Многие многошаговые методы различного порядка точности можно конструировать с помощью квадратурного способа (т.е. с использованием эквивалентного интегрального уравнения).

Решение дифференциального уравнения $y' = f(x, y)$ удовлетворяет интегральному соотношению:

$$y_{k+1} = y_k + \int_{x_k}^{x_{k+1}} f(x, y(x))dx$$

Если решение задачи Коши получено в узлах вплоть до k-го, то можно аппроксимировать подынтегральную функцию, например: интерполяционным многочленом какой-либо степени. Вычислив интеграл от построенного многочлена на отрезке $[x_k, x_{k+1}]$ получим ту или иную формулу Адамса. В частности, если использовать многочлен нулевой степени (то есть заменить подынтегральную функцию ее значением на левом конце отрезка в точке x_k), то получим явный метод Эйлера. Если проделать то же самое, но подынтегральную функцию аппроксимировать значением на правом конце в точке x_{k+1} , то получим неявный метод Эйлера.

При использовании интерполяционного многочлена 3-ей степени построенного по значениям подынтегральной функции в последних четырех узлах получим метод Адамса четвертого порядка точности:

$$y_{k+1} = y_k + \frac{h}{24}(55f_k - 59f_{k-1} + 37f_{k-2} - 9f_{k-3}),$$

где f_k значение подынтегральной функции в узле x_k.

Метод Адамса как и все многошаговые методы не является самостартующим, то есть для того, что бы использовать метод Адамса необходимо иметь решения в первых четырех узлах. В узле x_0 решение y_0 известно из начальных условий, а в других трех узлах x_1, x_2, x_3 решения y_1, y_2, y_3 можно получить с помощью подходящего одношагового метода, например: метода Рунге-Кутты четвертого порядка.

Пример

Реализовать методы Адамса 4-го порядка в виде программы, задавая в качестве входных данных шаг сетки h. С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

№	Задача Коши	Точное решение
1	$y'' + y - sin3x = 0,$ $y(0) = 1,$ $y'(0) = 1,$ $x \in [0, 1], h = 0.1$	$y = cosx + \frac{11}{8}sinx - \frac{sin3x}{8}$

Решение.

Для нахождения решения в первых узлах беем использовать результаты решения этой задачи методом Рунге-Кутты четвертого порядка.

x	y	$f(x_k, y_k)$	y_{src}	ϵ_k	Romberg's method error
0	1	1	1.0	0.0	0.0
0.1	1.09534	0.91005	1.09534	1e-06	5e-07
0.2	1.18266	0.83942	1.18266	1e-06	1.1e-06
0.3	1.26376	0.78496	1.26376	1e-06	1.5e-06
0.4	1.34009	0.74099	1.34001	8.5e-05	0.0094652
0.5	1.41221	0.7003	1.41211	0.000109	0.0183401
0.6	1.48008	0.65468	1.47999	8.9e-05	0.0272846
0.7	1.54275	0.59573	1.54274	1e-05	0.0361916
0.8	1.59851	0.51582	1.59864	0.000133	0.0448635
0.9	1.64493	0.40885	1.64526	0.00033	0.0531971
1.0	1.67911	0.27096	1.67968	0.000572	0.0610962

2 Численные методы решение краевой задачи для ОДУ. Примером краевой задачи является двухточечная краевая задача для обыкновенного дифференциального уравнения второго порядка.

$$y'' = f(x, y, y')$$

с граничными условиями, заданными на концах отрезка [a ,b].

$$y(a) = y_0$$
$$y(b) = y_1$$

Следует найти такое решение $y(x)$ на этом отрезке, которое принимает на концах отрезка значения y_0, y_1. Если функция $f(x, y, y')$ линейна по аргументам y, y', то задача - линейная краевая задача, в противном случае – нелинейная.

Кроме граничных условий называемых граничными условиями первого рода, используются еще условия на производные от решения на концах - граничные условия второго рода:

$$y'(a) = \tilde{y}_0$$
$$y'(b) = \tilde{y}_1$$

или линейная комбинация решений и производных - граничные условия третьего рода:

$$\alpha y(a) + \beta y'(a) = \tilde{y}_0,$$
$$\delta y(b) + \gamma y'(b) = \tilde{y}_1$$

$\alpha, \beta, \delta, \gamma$ – такие числа, что $|\alpha| + |\beta| \neq 0, |\delta| + |\gamma| \neq 0$

Метод стрельбы

Суть метода заключена в многократном решении задачи Коши для приближенного нахождения решения краевой задачи.

Пусть надо решить краевую задачу на отрезке . Вместо исходной задачи формулируется задача Коши с уравнением и с начальными условиями

$$y(a) = y_0$$
$$y'(b) = \eta$$

где η - некоторое значение тангенса угла наклона касательной к решению в точке $x = a$.

Положим сначала некоторое начальное значение параметру $\eta = \eta_0$, после чего решим каким либо методом задачу Коши. Пусть $y = y_0(x, y_0, \eta_0)$ решение этой задачи на интервале [a,b], тогда сравнивая значение функции $y_0(b, y_0, \eta_0)$ со значением y_1 в правом конце отрезка можно получить информацию для корректировки угла наклона касательной к решению в левом конце отрезка. Решая задачу Коши для нового значения $\eta = \eta_1$, получим другое решение со значением $y_1(b, y_0, \eta_1)$ на правом конце. Таким образом, значение решения на правом конце $y(b, y_0, \eta)$ будет являться функцией одной переменной η. Задачу можно сформулировать таким образом: требуется найти такое значение переменной η^* , чтобы решение $y(b, y_0, \eta^*)$ в правом конце отрезка совпало со значением y_1. Другими словами решение исходной задачи эквивалентно нахождению корня уравнения

$$\Phi(\eta) = 0,$$

где $\Phi(\eta) = y(b, y_0, \eta) - y_1$

Уравнение является “алгоритмическим” уравнением, так как левая часть его задается с помощью алгоритма численного решения соответствующей задачи Коши. Но методы решения уравнения аналогичны методам решения нелинейных уравнений, изложенным в разделе 2. Следует заметить, что так как невозможно вычислить производную функции $\Phi(\eta)$, то вместо метода Ньютона следует использовать метод секущих, в котором производная от функции заменена ее разностным аналогом. Данный разностный аналог легко вычисляется по двум приближениям, например $\eta_k \eta_{k+1}$.

Следующее значение искомого корня определяется по соотношению

$$\eta_{j+2} = \eta_{j+1} + \frac{n_{j+1} - n_j}{\Phi(\eta_{j+1}) - \Phi(\eta_j)} \Phi(\eta_{j+1})$$

Итерации по формуле выполняются до удовлетворения заданной точности.

Пример

Реализовать метод стрельбы решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

№	Задача Коши	Точное решение
1	$xy'' + 2y' - xy = 0,$ $y'(1) = 0,$ $1.5y(2) + y'(2) = e^2,$	$y(x) = \frac{e^x}{x}$

Решение

Заменой переменных $z = y'$сведем дифференциальное уравнение второго порядка к системе двух дифференциальных уравнений первого порядка

$$y' = z,$$
$$z' = \frac{(xy - 2y')}{x}$$

Задачу Коши для системы с начальными условиями на левом конце $y(1) = \eta, y'(1) = 0$ будем решать методом Рунге — Кутты четвертого порядка точности с шагом h = 0.1 до удовлетворения на правом конце условия

$$|1.5 * y(1, \eta_k, 0.0) + y'(1, \eta_k, 0.0)) - e^2| <= \epsilon = 0.0001$$

Примем в качестве первых двух значений параметра η следующие: $\eta_0 = 1.0, \eta_1 = 0.8$. Дважды решив задачу Коши с этими параметрами методом Рунге – Кутты с шагом

h = 0.1, получим два решения:

$$y(1, \eta_0, 0.0) = 1.359142328788921, y(1, \eta_1, 0.0) = 1.087313863031137$$

Вычислим новое приближение параметра η по формуле:

$$\eta_2 = 0.8 - \frac{(0.8 - 1)}{(2.14787 - 1.39258)} * (2.14787 - 7.38906) = 2.75214$$

Решая задачу Коши с параметром η_2, получим решение $y(1, \eta_2, 0.0) = 3.7405466629205852$
Остальные вычисления заносим в таблицу:

j	η_j	y	$\Phi(\eta_j)$
0	1	1.359142328788921	4.704213529419357
1	0.8	1.087313863031137	5.241182043321615
2	2.752137567706865	3.7405466629205852	3.552713678800501e-15

x	y	y_{src}	ϵ_k	Romberg's method error
1	2.75214	2.71828	0.03386	0.0113231
1.1	2.76508	2.73106	0.03402	0.01137563
1.2	2.80123	2.76676	0.03446	0.01152403
1.3	2.85769	2.82254	0.03516	0.01175621
1.4	2.93265	2.89657	0.03608	0.01206457
1.5	3.02501	2.98779	0.03722	0.01244456
1.6	3.13421	3.09565	0.03856	0.01289386
1.7	3.26008	3.21997	0.04011	0.01341178
1.8	3.40278	3.36092	0.04186	0.01399894
1.9	3.56272	3.51889	0.04383	0.01465704
2.0	3.74055	3.69453	0.04602	0.0153887

Пример

Реализовать конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

№	Задача Коши	Точное решение
1	$xy'' + 2y' - xy = 0,$ $y'(1) = 0,$ $1.5y(2) + y'(2) = e^2,$	$y(x) = \frac{e^x}{x}$

Решение

Здесь

$$p(x) = \frac{2}{x}, q(x) = -1, f(x) = 0, N = 10$$

$$x_0 = 1, x_1 = 1.1, x_2 = 1.2, x_3 = 1.3, x_4 = 1.4, x_5 = 1.5$$

$$x_6 = 1.6, x_7 = 1.7, x_8 = 1.8, x_9 = 1.9, x_{10} = 2.0$$

С помощью группировки слагаемых, приведения подобных членов и подстановки значений $_k$ и с учетом граничного условия получим систему линейных алгебраических уравнений:

$$-11.166666666666664y_0 + 11.111111111111109y_1 = 0.0$$

$$90.90909090909089y_0 + -200.99999999999997y_1 + 109.09090909090908y_2 = 0$$

$$91.66666666666666y_1 + -200.99999999999997y_2 + 108.33333333333331y_3 = 0$$

$$92.30769230769229y_2 + -200.99999999999997y_3 + 107.69230769230768y_4 = 0$$

$$92.85714285714285y_3 + -200.99999999999997y_4 + 107.14285714285712y_5 = 0$$

$$93.33333333333331y_4 + -200.99999999999997y_5 + 106.66666666666666y_6 = 0$$

$$93.74999999999999y_5 + -200.99999999999997y_6 + 106.24999999999999y_7 = 0$$

$$94.11764705882352y_6 + -200.99999999999997y_7 + 105.88235294117645y_8 = 0$$

$$94.44444444444443y_7 + -200.99999999999997y_8 + 105.55555555555554y_9 = 0$$

$$94.73684210526315y_8 + -200.99999999999997y_9 + 105.26315789473682y_10 = 0$$

$$-9.523809523809524y_9 + 11.071428571428571y_10 = 7.3890560989306495$$

В данной трехдиагональной системе выполнено условие преобладания диагональных элементов, и можно использовать метод прогонки. В результате решения системы методом прогонки получим следующие значения:

$$y_0 = 2.71433$$

$$y_1 = 2.7279$$

$$y_2 = 2.76421$$

$$y_3 = 2.82046$$

$$y_4 = 2.89486$$

$$y_5 = 2.98636$$

$$y_6 = 3.09441$$

$$y_7 = 3.21888$$

$$y_8 = 3.35992$$

$$y_9 = 3.51795$$

$$y_10 = 3.69359$$

Таблица с результатами:

x	y	y_{src}	ϵ_k	Romberg's method error
1	2.71433	2.71828	0.00395	2.53e-06
1.1	2.7279	2.73106	0.00316	1.94e-06
1.2	2.76421	2.76676	0.00255	1.48e-06
1.3	2.82046	2.82254	0.00208	1.12e-06
1.4	2.89486	2.89657	0.00171	8.3e-07
1.5	2.98636	2.98779	0.00144	6e-07
1.6	3.09441	3.09565	0.00123	4.1e-07
1.7	3.21888	3.21997	0.00109	2.6e-07
1.8	3.35992	3.36092	0.00099	1.4e-07
1.9	3.51795	3.51889	0.00094	4e-08
2.0	3.69359	3.69453	0.00094	3e-08

2 Исходный код

1 Метод Эйлера, Рунге-Кутты и Адамса 4-го порядка

```
import argparse
import numpy as np
from prettytable import PrettyTable


def foo(x, y, y1):
    # return (y + x) ** 2
    # return (2 * x * y1) / (x ** 2 + 1)
    return np.sin(3 * x) - y


def orig_foo(x):
    # return np.tan(x) - x
    # return x ** 3 + 3 * x + 1
    return np.cos(x) + 11 / 8 * np.sin(x) - np.sin(3 * x) / 8


def euler(f, xa, xb, ya, y1a, h, fl=None):
    if fl:
        fl.write(f'Euler:\n\n')
        fl.write(f'z = y\' \n'
                 f'z\' = sin(3x) - y \n'
                 f'[{xa}, {xb}]\n'
                 f'x0 = {xa}, y0 = {ya}, z0 = {y1a}\n\n')
    n = int((xb - xa) / h)
    x = xa
    y = ya
    x_res = [x]
    y_res = [y]
    deltaYk = []
    y = ya
    y1 = y1a
    for i in range(n):
        if i < 2 and fl:
            fl.write(f'x{i + 1} = {x + h}\n'
                     f'z{i + 1} = z{i} + hf(x{i},y{i}, y\'{i}) = '
                     f'{y1} + {h} * ({np.sin(3 * x)} - {y}) = {y1 + h * f(x, y, y1)}\n'
                     f'y{i + 1} = y{i} + hz = {y + h * (y1 + h * f(x, y, y1))}\n\n')
        deltaYk.append(h * f(x, y, y1))
        y1 += h * f(x, y, y1)
        y += h * y1
        # y += h * f(x, y, y1)
        x += h
        x_res.append(x)
        y_res.append(y)
    deltaYk.append('-')
```

```
    x_res = [round(x, 5) for x in x_res]
    return x_res, y_res, deltaYk


def runge_kutta(f, xa, xb, ya, y1a, h, fi=None):
    n = int((xb - xa) / h)
    x = xa
    y = ya
    z = y1a
    x_res = [x]
    y_res = [y]
    z_res = [z]
    deltaY = []
    deltaZ = []
    for i in range(1, n + 1):
        k1 = h * z
        l1 = h * f(x, y, z)
        k2 = h * (z + 0.5 * l1)
        l2 = h * f(x + 0.5 * h, y + 0.5 * k1, z + 0.5 * l1)
        k3 = h * (z + 0.5 * l2)
        l3 = h * f(x + 0.5 * h, y + 0.5 * k2, z + 0.5 * l2)
        k4 = h * (z + l3)
        l4 = h * f(x + h, y + k3, z + l3)
        x = xa + i * h
        deltaY.append((k1 + 2 * k2 + 2 * k3 + k4) / 6)
        deltaZ.append((l1 + 2 * l2 + 2 * l3 + l4) / 6)

        y += (k1 + 2 * k2 + 2 * k3 + k4) / 6
        z += (l1 + 2 * l2 + 2 * l3 + l4) / 6
        if fi and i == 1:
            fi.write(f'\nRunge-Kutt:\n\n')
            fi.write(f"K_1 = {k1}\n"
                     f"L_1 = {l1}\n"
                     f"K_2 = {k2}\n"
                     f"L_2 = {l2}\n"
                     f"K_3 = {k3}\n"
                     f"L_3 = {l3}\n"
                     f"K_4 = {k4}\n"
                     f"L_4 = {l4}\n\n")
            fi.write(f"deltaY_0 = {deltaY[0]}\n"
                     f"deltaZ_0 = {deltaZ[0]}\n\n")
            fi.write(f"y1 = {y}\n")
            fi.write(f"z1 = {z}\n\n")
        x_res.append(x)
        y_res.append(y)
        z_res.append(z)
    deltaY.append(0)
    deltaZ.append(0)
    x_res = [round(x, 5) for x in x_res]
```

```
96      return x_res, y_res, z_res, deltaY, deltaZ
97
98
99  def adams(f, x, y, h, n, z):
100     z = z[:4] + [0] * (len(z) - 4)
101     for i in range(3, n):
102         z[i + 1] = z[i] + h / 24 * (55 * f(x[i], y[i], z[i]) - \
103                                    59 * f(x[i - 1], y[i - 1], z[i - 1]) + \
104                                    37 * f(x[i - 2], y[i - 2], z[i - 2]) - \
105                                    9 * f(x[i - 3], y[i - 3], z[i - 3]))
106         tmp = y[i] + h / 24 * (55 * z[i] - 59 * z[i - 1] + \
107                                37 * z[i - 2] - 9 * z[i - 3])
108         x.append(x[-1] + h)
109         y.append(tmp)
110     x = [round(i, 2) for i in x]
111     return x, y, z
112
113
114 def print_result_table(f, name, res, runge_y, p):
115     if 'Euler' == name:
116         table = PrettyTable(['x', 'y', 'delta_Yk', 'y_src', 'eps_k',
117                              'Romberg\'s method error'])
118         for x, y, delta, yr in zip(*res, runge_y):
119             tmp = orig_foo(x)
120             table.add_row([round(x, 8), round(y, 8), delta, round(tmp, 8), abs(y - tmp)
                ,
121                            abs(y + abs(y - yr) / (0.5 ** p - 1) - tmp)])
122         f.write(f'\n{str(table)}\n')
123     elif 'Runge-Kutta' == name:
124
125         table = PrettyTable(['x', 'y', 'z', 'delta_Yk', 'delta_Zk', 'y_src', 'eps_k',
126                              'Romberg\'s method error'])
127         for x, y, z, deltaY, deltaZ, yr in zip(*res, runge_y):
128             tmp = orig_foo(x)
129             table.add_row([round(x, 5), round(y, 5), round(z, 5), round(deltaY, 5),
130                            round(deltaZ, 5), round(tmp, 5), abs(round(y - tmp, 6)),
131                            abs(y + abs(y - yr) / (0.5 ** p - 1) - tmp)])
132         f.write(f'\n{str(table)}\n')
133     elif 'Adams' == name:
134         f.write(f'\nAdams:\n')
135         table = PrettyTable(['x_k', 'y_k', 'f(x_k, y_k)', 'y_src', 'eps_k',
136                              'Romberg\'s method error'])
137         for x, y, z, yr in zip(*res, runge_y):
138             tmp = orig_foo(x)
139             table.add_row([round(x, 5), round(y, 5), round(z, 5), round(tmp, 5), abs(
                    round(y - tmp, 6)),
140                            abs(round(y + abs(y - yr) / (0.5 ** p - 1) - tmp, 7))])
141         f.write(f'\n{str(table)}\n')
142
```

```
143
144 def main():
145     parser = argparse.ArgumentParser()
146     parser.add_argument('--output', required=True, help='File for answer')
147     parser.add_argument('--h', required=True, help='Step', type=float)
148     args = parser.parse_args()
149
150     # a = 0
151     # b = 0.5
152     # y0 = 0
153     # y10 = 0
154     a = 0
155     b = 1
156     y0 = 1
157     y10 = 1
158     step = args.h
159     f = open(args.output, 'w')
160
161     res1 = euler(foo, a, b, y0, y10, step, f)
162     res1_half_h = euler(foo, a, b, y0, y10, step / 2)
163     h_half = [y for x, y in zip(res1_half_h[0], res1_half_h[1]) if x in res1[0]]
164     print_result_table(f, 'Euler', res1, h_half, 1)
165
166     res2 = runge_kutta(foo, a, b, y0, y10, step, f)
167     z = res2[2]
168     res2_half_h = runge_kutta(foo, a, b, y0, y10, step / 2)
169     z_half_h = res2_half_h[2]
170     h_half = [y for x, y in zip(res2_half_h[0], res2_half_h[1]) if x in res2[0]]
171     print_result_table(f, 'Runge-Kutta', res2, h_half, 4)
172
173     res3 = adams(foo, res2[0][:4], res2[1][:4], step, int((b - a) / step), z)
174     res3_half_h = adams(foo, res2[0][:4], res2[1][:4], step / 2,
175                         int((b - a) / (step / 2)), z_half_h)
176     h_half = [y for x, y in zip(res3_half_h[0], res3_half_h[1]) if x in res3[0]]
177     print_result_table(f, 'Adams', res3, h_half, 4)
178
179     f.close()
180
181
182 if __name__ == "__main__":
183     main()
```

2 Численные методы решение краевой задачи для ОДУ.

```
1 import argparse
2 import numpy as np
3 from prettytable import PrettyTable
4
5
6 def func(x, y, y1):
```

```
7       # return np.e ** x + np.sin(y)
8       return (x * y - 2 * y1) / x
9
10
11  def orig_func(x):
12      return np.exp(x) / x
13
14
15  def p(x):
16      return 2 / x
17
18
19  def q(x):
20      return -1
21
22
23  def f(x):
24      return 0
25
26
27  def tma(a, b, c, d, shape):
28      p = [-c[0] / b[0]]
29      q = [d[0] / b[0]]
30      x = [0] * (shape + 1)
31      for i in range(1, shape):
32          p.append(-c[i] / (b[i] + a[i] * p[i - 1]))
33          q.append((d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]))
34      for i in reversed(range(shape)):
35          x[i] = p[i] * x[i + 1] + q[i]
36      return x[:-1]
37
38
39  def runge_kutta(f, xa, xb, ya, y1a, h):
40      n = int((xb - xa) / h)
41      x = xa
42      y = ya
43      z = y1a
44      x_res = [x]
45      y_res = [y]
46      z_res = [z]
47      for i in range(1, n + 1):
48          k1 = h * z
49          l1 = h * f(x, y, z)
50          k2 = h * (z + 0.5 * l1)
51          l2 = h * f(x + 0.5 * h, y + 0.5 * k1, z + 0.5 * l1)
52          k3 = h * (z + 0.5 * l2)
53          l3 = h * f(x + 0.5 * h, y + 0.5 * k2, z + 0.5 * l2)
54          k4 = h * (z + l3)
55          l4 = h * f(x + h, y + k3, z + l3)
```

```
        x = xa + i * h
        y += (k1 + 2 * k2 + 2 * k3 + k4) / 6
        z += (l1 + 2 * l2 + 2 * l3 + l4) / 6
        x_res.append(x)
        y_res.append(y)
        z_res.append(z)
    return (x_res, y_res), z_res


def der_one(xi, yi, x):
    i = 0
    while xi[i + 1] < x - 1e-7:
        i += 1
    return (yi[i + 1] - yi[i]) / (xi[i + 1] - xi[i])


def next_n(cur_n, prev_n, ans_cur, ans_prev, alpha1, beta1, B, b, fi=None):
    num1 = beta1 * der_one(ans_cur[0], ans_cur[1], b)
    num2 = beta1 * der_one(ans_prev[0], ans_prev[1], b)
    num3 = alpha1 * ans_prev[1][len(ans_prev[0]) - 1]
    num4 = alpha1 * ans_cur[1][len(ans_cur[0]) - 1] + num1 - B
    num5 = alpha1 * ans_cur[1][len(ans_cur[0]) - 1] + num1 - num3 - num2
    if fi:
        fi.write(f'{round(cur_n, 5)} - ({round(cur_n, 5)} - {round(prev_n, 5)}) / '
                 f'({round(alpha1 * ans_cur[1][len(ans_cur[0]) - 1] + num1, 5)} - {round
                     (num3 - num2, 5)}) * '
                 f'({round(alpha1 * ans_cur[1][len(ans_cur[0]) - 1] + num1, 5)} - {round
                     (B, 5)})'
                 f' = {round(cur_n - num4 * (cur_n - prev_n) / num5, 5)}\n')
    return cur_n - num4 * (cur_n - prev_n) / num5


def shooting_method(a, b, h, eps, f, alpha0, alpha1, beta0, beta1, A, B, fi=None):
    table = PrettyTable(['j', 'n_j', 'y', '|Phi(n_j)|'])

    n_prev = 1
    n_cur = 0.8
    if fi:
        fi.write(f'|{alpha1} * y({a}, n_k, {(A - alpha0 * n_prev) / beta0}) '
                 f'+ {beta1} * y\'({a}, n_k, {(A - alpha0 * n_prev) / beta0})) - {B}| <=
                      eps\n')
        fi.write(f'eps = {eps}\n')
    ans_prev = runge_kutta(f, a, b, n_prev, (A - alpha0 * n_prev) / beta0, h)[0]
    ans_cur = runge_kutta(f, a, b, n_cur, (A - alpha0 * n_cur) / beta0, h)[0]
    table.add_row([0, n_prev, ans_prev[1][-1], abs(alpha1 * ans_prev[1][len(ans_prev
        [0]) - 1] +
                                                   beta1 * der_one(ans_prev[0], ans_prev
                                                       [1], b) - B)])
    table.add_row([1, n_cur, ans_cur[1][-1], abs(alpha1 * ans_cur[1][len(ans_cur[0]) -
```

```
        1] +
100                                             beta1 * der_one(ans_cur[0], ans_cur[1], b)
                                                    - B)])
101     i = 2
102     while abs(alpha1 * ans_cur[1][len(ans_cur[0]) - 1] +
103               beta1 * der_one(ans_cur[0], ans_cur[1], b) - B) > eps:
104         if i == 2 or i == 3:
105             n = next_n(n_cur, n_prev, ans_cur, ans_prev, alpha1, beta1, B, b, fi)
106         else:
107             n = next_n(n_cur, n_prev, ans_cur, ans_prev, alpha1, beta1, B, b)
108         n_prev = n_cur
109         n_cur = n
110         ans_prev = ans_cur
111         ans_cur = runge_kutta(f, a, b, n_cur, (A - alpha0 * n_cur) / beta0, h)[0]
112         table.add_row([i, n_cur, ans_cur[1][-1], abs(alpha1 * ans_cur[1][len(ans_cur
                [0]) - 1] + \
113                                                 beta1 * der_one(ans_cur[0], ans_cur[1],
                                                        b) - B)])
114         i += 1
115     if fi:
116         fi.write(f'\n{str(table)}\n')
117     return ans_cur
118
119
120 def finite_difference_method(a1, b1, h, alpha_0, alpha_1, beta_0, beta_1, A, B, fi=
        None):
121     x = [a1]
122     a = []
123     b = []
124     c = []
125     d = []
126     n = round((b1 - a1) / h)
127     a.append(0)
128     b.append(-2 / (h * (2 - p(a1) * h)) + q(a1) * h /
129              (2 - p(a1) * h) + alpha_0 / beta_0)
130     c.append(2 / (h * (2 - p(a1) * h)))
131     d.append(A / beta_0 + h * f(a1) / (2 - p(a1) * h))
132     x.append(x[0] + h)
133     if fi:
134         fi.write(f'{b[0]}y_0 + {c[0]}y_1 = {d[0]}\n')
135
136     for i in range(1, n):
137         a.append(1 / h ** 2 - p(x[i]) / (2 * h))
138         b.append(-2 / h ** 2 + q(x[i]))
139         c.append(1 / h ** 2 + p(x[i]) / (2 * h))
140         d.append(f(x[i]))
141         x.append(x[i] + h)
142         if fi:
143             fi.write(f'{a[i]}y_{i - 1} + {b[i]}y_{i} + {c[i]}y_{i+1} = {d[i]}\n')
```

```
144     a.append(-2 / (h * (2 + p(x[n]) * h)))
145     b.append(2 / (h * (2 + p(x[n]) * h)) - q(x[n]) * h /
146             (2 + p(x[n]) * h) + alpha_1 / beta_1)
147     c.append(0)
148     d.append(B / beta_1 - h * f(x[n]) / (2 + p(x[n]) * h))
149     if fi:
150         fi.write(f'{a[-1]}y_{len(a) - 2} + {b[-1]}y_{len(a) - 1} = {d[-1]}\n')
151     y = tma(a, b, c, d, len(a))
152     # print(a)
153     # print(b)
154     # print(c)
155     # print(d)
156     return x, y
157
158
159 def print_result_table(f, name, res, runge_y, p):
160     f.write(f'\n{name}\n')
161     table = PrettyTable(['x', 'y', 'y_src', 'eps_k',
162                          'Romberg\'s method error'])
163     for x, y, yr in zip(*res, runge_y):
164         tmp = orig_func(x)
165         table.add_row([round(x, 2), round(y, 5), round(tmp, 5), abs(round(y - tmp, 5)),
166                        abs(round(y + (y - yr) / (0.5 ** p - 1) - tmp, 8))])
167     f.write(f'\n{str(table)}\n')
168
169
170 def main():
171     parser = argparse.ArgumentParser()
172     parser.add_argument('--output', required=True, help='File for answer')
173     parser.add_argument('--h', required=True, help='Step', type=float)
174     parser.add_argument('--eps', type=float, help='Epsilon', default=1e-5)
175     args = parser.parse_args()
176
177     a = 1
178     b = 2
179     alpha0 = 0
180     alpha1 = 1.5
181     beta0 = 1
182     beta1 = 1
183     y0 = 0
184     y10 = np.e ** 2
185     # a = 0
186     # b = 1
187     # alpha0 = 1
188     # alpha1 = 1
189     # beta0 = 0
190     # beta1 = 0
191     # y0 = 1
192     # y10 = 2
```

```
193     step = args.h
194     eps = args.eps
195     f = open(args.output, 'w')
196 
197     res1 = shooting_method(a, b, step, eps, func, alpha0, alpha1,
198                            beta0, beta1, y0, y10, f)
199     res2 = shooting_method(a, b, step / 2, eps, func, alpha0, alpha1,
200                            beta0, beta1, y0, y10)
201     h_half = [y for x, y in zip(res2[0], res2[1]) if x in res1[0]]
202     print_result_table(f, 'Shooting method', res1, h_half, 2)
203 
204     res1 = finite_difference_method(a, b, step, alpha0, alpha1,
205                                     beta0, beta1, y0, y10, f)
206     res2 = finite_difference_method(a, b, step / 2, alpha0, alpha1,
207                                     beta0, beta1, y0, y10)
208     h_half = [y for x, y in zip(res2[0], res2[1]) if x in res1[0]]
209     print_result_table(f, 'Finite difference method', res1, h_half, 2)
210 
211     f.close()
212 
213 
214 if __name__ == "__main__":
215     main()
```

3 Вывод программы

1

Входные данные: Имя файла для результата и шаг h.

Выходные данные: В выходном файле таблицы со значениями.

```
art@mars:~/study/NM/lab_4/p_1 python3 Lw4_1.py --output output --h 0.1
art@mars:~/study/NM/lab_4/p_1 cat output
Euler:

z = y'
z'= sin(3x) -y
[0,1]
x0 = 0,y0 = 1,z0 = 1

x1 = 0.1
z1 = z0 + hf(x0,y0,y'0) = 1 + 0.1 * (0.0 -1) = 0.9
y1 = y0 + hz = 1.09

x2 = 0.2
z2 = z1 + hf(x1,y1,y'1) = 0.9 + 0.1 * (0.2955202066613396 -1.09) = 0.820552020666134
y2 = y1 + hz = 1.1720552020666135


+-----+------------+-----------------------+------------+-----------------------+
|  x  |     y      |        delta_Yk       |   y_src    |         eps_k         |
+-----+------------+-----------------------+------------+-----------------------+
|  0  |     1      |          -0.1         |    1.0     |          0.0          |
| 0.1 |    1.09    |  -0.07944797933386605 | 1.09533509 | 0.0053350873347468575
|
| 0.2 | 1.1720552  |  -0.0607412728671578  | 1.1826566  |   0.0106013964434577
|
| 0.3 | 1.24803628 |  -0.04647093672190277 | 1.26376091 |  0.015724632735001354
|
| 0.4 | 1.31937026 |  -0.03873311719869922 | 1.34000633 |  0.020636070977157805
|
| 0.5 | 1.38683093 |  -0.03893359407380016 | 1.4121058  |  0.025274876803588864
|
| 0.6 | 1.45039824 | -0.047655060644431826 | 1.47998806 |  0.029589824645564144
|
| 0.7 | 1.50920004 |  -0.06459906745896539 | 1.54274034 |  0.03354029516667656
```

```
|
| 0.8 | 1.56154194 |  -0.08860787578574947 | 1.59863844 |  0.037096498356469665
|
| 0.9 | 1.60502305 |  -0.11776431677663582 | 1.64526198 |  0.040238935979036716
|
| 1.0 | 1.63672773 |           -          | 1.67968491 |  0.042957183057446224
|
+-----+------------+-----------------------+------------+-----------------------+
------------------------+
Romberg's method error |
------------------------+
0.0           |
0.01084477799711503   |
0.02154246256086423   |
0.03193192142376611   |
0.041865460344849614  |
0.05121438933729516   |
0.05987308610652309   |
0.06776126650175018   |
0.07482432950978124   |
0.0810318131883554   |
0.08637416289208999   |
------------------------+

Runge-Kutt:

K_1 = 0.1
L_1 = -0.1
K_2 = 0.095
L_2 = -0.09005618675264009
K_3 = 0.095497190662368
L_3 = -0.08980618675264009
K_4 = 0.091019381324736
L_4 = -0.07999769840010285

deltaY_0 = 0.09533562710824534
deltaZ_0 = -0.0899537409017772

y1 = 1.0953356271082453
z1 = 0.9100462590982228
```

x	y	z	delta_Yk	delta_Zk	y_src	eps_k
0	1	1	0.09534	-0.08995	1.0	0.0
0.1	1.09534	0.91005	0.08732	-0.07062	1.09534	1e-06
0.2	1.18266	0.83942	0.0811	-0.05446	1.18266	1e-06
0.3	1.26376	0.78496	0.07625	-0.04381	1.26376	1e-06
0.4	1.34001	0.74116	0.0721	-0.04043	1.34001	2e-06
0.5	1.41211	0.70073	0.06788	-0.04533	1.41211	2e-06
0.6	1.47999	0.6554	0.06275	-0.05864	1.47999	2e-06
0.7	1.54274	0.59676	0.0559	-0.07962	1.54274	2e-06
0.8	1.59864	0.51714	0.04662	-0.10672	1.59864	2e-06
0.9	1.64526	0.41042	0.03442	-0.13772	1.64526	1e-06
1.0	1.67969	0.27269	0	0	1.67968	1e-06

Romberg's method error
0.0
3.4429792350465505e-11
1.6584467132929603e-10
5.652300849590119e-10
1.1037109004519152e-09
1.7029053722694698e-09
2.2738670946154116e-09
2.7259083879016544e-09
2.9756614949860705e-09
2.955536704263295e-09
2.620814898435242e-09

Adams:

x_k	y_k	f(x_k,y_k)	y_src	eps_k	Romberg's method error
0	1	1	1.0	0.0	0.0
0.1	1.09534	0.91005	1.09534	1e-06	5e-07
0.2	1.18266	0.83942	1.18266	1e-06	1.1e-06

```
| 0.3 | 1.26376 |   0.78496   | 1.26376 |  1e-06   |        1.5e-06         |
| 0.4 | 1.34009 |   0.74099   | 1.34001 | 8.5e-05  |       0.0094652        |
| 0.5 | 1.41221 |    0.7003   | 1.41211 | 0.000109 |       0.0183401        |
| 0.6 | 1.48008 |   0.65468   | 1.47999 | 8.9e-05  |       0.0272846        |
| 0.7 | 1.54275 |   0.59573   | 1.54274 |  1e-05   |       0.0361916        |
| 0.8 | 1.59851 |   0.51582   | 1.59864 | 0.000133 |       0.0448635        |
| 0.9 | 1.64493 |   0.40885   | 1.64526 | 0.00033  |       0.0531971        |
| 1.0 | 1.67911 |   0.27096   | 1.67968 | 0.000572 |       0.0610962        |
+-----+---------+-------------+---------+----------+------------------------+
art@mars:~/study/NM/lab_4/p_1
```

2

Входные данные: Имя файла для результата и шаг h.

Выходные данные: В выходном файле таблицы со значениями.

```
art@mars:~/study/NM/lab_4/p_2 python3 Lw4_2.py --output output --h 0.1
art@mars:~/study/NM/lab_4/p_2 cat output


Shooting method

|1.5 * y(1,n_k,0.0) + 1 * y'(1,n_k,0.0)) -7.3890560989306495| <= eps
eps = 1e-05
0.8 -(0.8 -1) / (2.14787 -1.39258) * (2.14787 -7.38906) = 2.75214

+---+------------------+--------------------+-----------------------+
| j |        n_j       |         y          |       |Phi(n_j)|      |
+---+------------------+--------------------+-----------------------+
| 0 |         1        | 1.359142328788921  |   4.704213529419357   |
| 1 |        0.8       | 1.087313863031137  |   5.241182043321615   |
| 2 | 2.752137567706865 | 3.7405466629205852 | 3.552713678800501e-15 |
+---+------------------+--------------------+-----------------------+

+-----+---------+---------+---------+------------------------+
|  x  |    y    |  y_src  |  eps_k  | Romberg's method error |
+-----+---------+---------+---------+------------------------+
|  1  | 2.75214 | 2.71828 | 0.03386 |       0.0113231        |
```

1.1	2.76508	2.73106	0.03402	0.01137563
1.2	2.80123	2.76676	0.03446	0.01152403
1.3	2.85769	2.82254	0.03516	0.01175621
1.4	2.93265	2.89657	0.03608	0.01206457
1.5	3.02501	2.98779	0.03722	0.01244456
1.6	3.13421	3.09565	0.03856	0.01289386
1.7	3.26008	3.21997	0.04011	0.01341178
1.8	3.40278	3.36092	0.04186	0.01399894
1.9	3.56272	3.51889	0.04383	0.01465704
2.0	3.74055	3.69453	0.04602	0.0153887
+-----+---------+---------+---------+------------------------+

Finite difference method

-11.166666666666664y_0 + 11.111111111111109y_1 = 0.0
90.90909090909089y_0 + -200.99999999999997y_1 + 109.09090909090908y_2 = 0
91.66666666666666y_1 + -200.99999999999997y_2 + 108.33333333333331y_3 = 0
92.30769230769229y_2 + -200.99999999999997y_3 + 107.69230769230768y_4 = 0
92.85714285714285y_3 + -200.99999999999997y_4 + 107.14285714285712y_5 = 0
93.33333333333331y_4 + -200.99999999999997y_5 + 106.66666666666666y_6 = 0
93.74999999999999y_5 + -200.99999999999997y_6 + 106.24999999999999y_7 = 0
94.11764705882352y_6 + -200.99999999999997y_7 + 105.88235294117645y_8 = 0
94.44444444444443y_7 + -200.99999999999997y_8 + 105.55555555555554y_9 = 0
94.73684210526315y_8 + -200.99999999999997y_9 + 105.26315789473682y_10 = 0
-9.523809523809524y_9 + 11.071428571428571y_10 = 7.3890560989306495

+-----+---------+---------+---------+------------------------+
| x | y | y_src | eps_k | Romberg's method error |
+-----+---------+---------+---------+------------------------+
1	2.71433	2.71828	0.00395	2.53e-06
1.1	2.7279	2.73106	0.00316	1.94e-06
1.2	2.76421	2.76676	0.00255	1.48e-06
1.3	2.82046	2.82254	0.00208	1.12e-06
1.4	2.89486	2.89657	0.00171	8.3e-07
1.5	2.98636	2.98779	0.00144	6e-07
1.6	3.09441	3.09565	0.00123	4.1e-07
1.7	3.21888	3.21997	0.00109	2.6e-07
1.8	3.35992	3.36092	0.00099	1.4e-07
1.9	3.51795	3.51889	0.00094	4e-08
2.0	3.69359	3.69453	0.00094	3e-08

+-----+---------+---------+---------+-----------------------+

4 Выводы

Благодаря этой лабораторной работе, я узнал, что можно быстро с помощью компьютера решать задачу Коши с помощью методов Эйлера, Рунге-Кутты и Адамса и решать краевую задачу с помощью метода стрельбы и конечно-разностного метода.

Список литературы

[1] *Численные методы. Учебник*
Пирумов Ульян Гайкович, Гидаспов Владимир Юрьевич
(ISBN 978-5-534-03141-6)