**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 4**

по курсу «Численные методы».

Тема: «Методы решения обыкновенных дифференциальных уравнений».

Студент: Шевчук

Группа: 80-304Б

Вариант: 13

Оценка:

Москва, 2019

Постановка задачи.

Реализовать методы для

решения задачи Коши:

1. метод Эйлера
2. метод Рунге-Кутты четвёртого порядка
3. метод Адамса четвёртого порядка

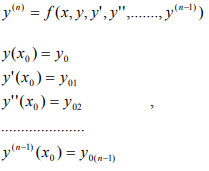
решения краевой задачи ОДУ:

1. метод стрельбы
2. метод конечных разностей

Описание методов.

**Задача Коши.**

Заданы обыкновенное дифференциально уравнение второго порядка *y’’ = f(x, y, y’)* и значения *y(a)* и *y’(a)* на границе *a* отрезка *[a,b].* Требуется найти сеточную функцию *y* с шагом *h* на отрезке *[a, b].* Сначала сведём ОДУ второго порядка к системе ОДУ первого порядка с помощью введения новых переменных



Далее мы можем представить систему в векторном виде и применять заданные методы, подставляя векторы в качестве переменных

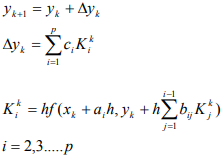


1. Редко используется на практике из-за невысокой точности. В случае небольшого шага разностной сетки h график функции и график касательной не успевают сильно разойтись друг от друга и можно в качестве значения решения в узле принять значение касательной , вместо значения неизвестного точного решения . При этом допускается погрешность. Повторяя все предыдущие рассуждения, получим значения в остальных точках.

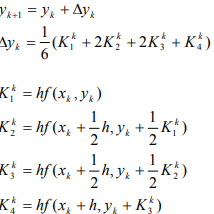
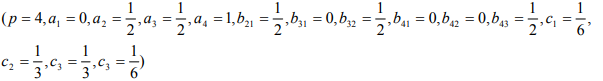
Итерационная формула . Погрешность на каждом шаге . Также контроль точности можно осуществлять методом Румбе-Ромберга или путём сравнения с точным результатом.



1. Часто используется на практике. Семейство явных методов Рунге-Кутты *р*-го порядка записывается в виде совокупности формул:



Для четвёртого порядка:



Контроль точности осуществляется методом Румбе-Ромберга или путём сравнения с точным результатом.

1. Решение дифференциального уравнения *y’ =f(x, y)* удовлетворяет интегральному соотношению



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, получим ту или иную формулу Адамса. Чтобы получить метод четвёртого порядка, будем использовать многочлен третей степени, получим итерационную формулу

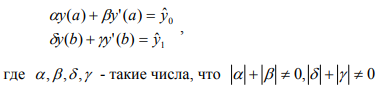


Контроль точности осуществляется методом Румбе-Ромберга или путём сравнения с точным результатом.

**Краевая задача.**

Заданы обыкновенное дифференциально уравнение второго порядка *y’’ = f(x, y, y’)* и условия на концах отрезка *[a, b]*

*.*



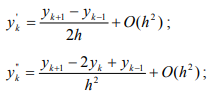
Сведём ОДУ второго порядка к системе ОДУ первого порядка так же, как сделали это для задачи Коши.

1. Суть метода заключена в многократном решении задачи Коши для приближенного нахождения решения краевой задачи. Вместо исходной задачи формулируется задача Коши, условие на правом конце заменяется на ещё одним условием на левом (y’(a) = η). Решается задача Коши, полученное решение на правом конце сравнивается с отброшенным краевым условием. По результатам сравнения делается корректировка η.

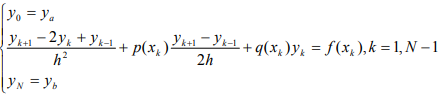
решение исходной задачи эквивалентно нахождению корня уравнения *Φ(η) = 0* , где *.* Так как невозможно вычислить производную функции Φ(η), будем пользоваться методом секущих. Следующее значение корня определяется соотношением . Условие окончания . Контроль точности осуществляется методом Румбе-Ромберга или путём сравнения с точным результатом.



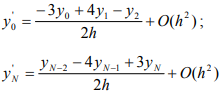
1. Введем разностную аппроксимацию производных следующим образом:



Подставим в исходное уравнение, получим:



Если мы используем аппроксимацию производных на краях с помощью односторонних разностей второго порядка, мы теряем трехдиагональную структуру матрицы коэффициентов, но сохраняем второй порядок аппроксимации:



Контроль точности осуществляется методом Румбе-Ромберга или путём сравнения с точным результатом.

Общая информация.

Данная работа состоит из двух модулей, которые позволяют решать различные задачи с краевыми условиями. Что касается технических деталей реализации, все программы написаны на языке python.

Запуск программы.

Запуск аналогично

Входные данные и результаты.

Вариант 1.

|  |  |
| --- | --- |
| **in1.txt** | **out1.txt**  Euler  +-----+-------+------------+---------------------+  | X | Y | Accurate Y | Runge eps |  +-----+-------+------------+---------------------+  | 0 | 1 | 1.0 | 0.0 |  | 0.1 | 1.17 | 1.185 | 0.0774999999999999 |  | 0.2 | 1.308 | 1.337 | 0.13104224304843148 |  | 0.3 | 1.413 | 1.455 | 0.15892155580910772 |  | 0.4 | 1.482 | 1.535 | 0.15913193006423598 |  | 0.5 | 1.512 | 1.575 | 0.12923414665020605 |  | 0.6 | 1.5 | 1.568 | 0.06610859979592787 |  | 0.7 | 1.441 | 1.511 | 0.03447852643972493 |  | 0.8 | 1.33 | 1.393 | 0.1786384806582224 |  | 0.9 | 1.155 | 1.201 | 0.37574799011737725 |  | 1.0 | 0.902 | 0.913 | 0.6411155173968406 |  +-----+-------+------------+---------------------+  Runge-Kutta  +-----+-------+------------+-----------------------+  | X | Y | Accurate Y | Runge eps |  +-----+-------+------------+-----------------------+  | 0 | 1 | 1.0 | 0.0 |  | 0.1 | 1.185 | 1.185 | 0.005896412161126507 |  | 0.2 | 1.337 | 1.337 | 0.010165200319027907 |  | 0.3 | 1.455 | 1.455 | 0.01265994548015567 |  | 0.4 | 1.535 | 1.535 | 0.013208071068462897 |  | 0.5 | 1.575 | 1.575 | 0.011598342280933609 |  | 0.6 | 1.568 | 1.568 | 0.007557864958379564 |  | 0.7 | 1.511 | 1.511 | 0.0007109101859945917 |  | 0.8 | 1.393 | 1.393 | 0.009496333023338025 |  | 0.9 | 1.201 | 1.201 | 0.023940120130651308 |  | 1.0 | 0.913 | 0.913 | 0.04411689079578318 |  +-----+-------+------------+-----------------------+  Adams  +-----+-------+------------+-----------------------+  | X | Y | Accurate Y | Runge eps |  +-----+-------+------------+-----------------------+  | 0 | 1 | 1.0 | 0.0 |  | 0.1 | 1.185 | 1.185 | 0.000092200143751793 |  | 0.2 | 1.337 | 1.337 | 0.000016754232436566 |  | 0.3 | 1.455 | 1.455 | 0.000022398707164023 |  | 0.4 | 1.535 | 1.535 | 0.0005511180574619819 |  | 0.5 | 1.575 | 1.575 | 0.0010296059730896681 |  | 0.6 | 1.569 | 1.568 | 0.004965602566728266 |  | 0.7 | 1.512 | 1.511 | 0.011698703008940603 |  | 0.8 | 1.395 | 1.393 | 0.02168098061857721 |  | 0.9 | 1.205 | 1.201 | 0.035770399280130594 |  | 1.0 | 0.921 | 0.913 | 0.055354181850205623 |  +-----+-------+------------+-----------------------+ |
| **in2.txt** | **out2.txt**  Shooting method  +------+--------------------+--------------------+----------------------+----------------------+  | X | Y | Accurate Y | |Y - Accurate Y| | Runge eps |  +------+--------------------+--------------------+----------------------+----------------------+  | 0 | 0.5367859760800582 | 0.5 | 0.03678597608005818 | 0.006167255016460575 |  | 0.03 | 0.5558572418772012 | 0.5190654460249184 | 0.03679179585228276 | 0.009372939973961714 |  | 0.05 | 0.5755832464006435 | 0.5387736998918138 | 0.03680954650882973 | 0.012743565529478307 |  | 0.08 | 0.5959826049218105 | 0.5591429350060964 | 0.03683966991571419 | 0.016284583008802866 |  | 0.1 | 0.6170743457311233 | 0.5801917305967077 | 0.0368826151344156 | 0.02000157304288282 |  | 0.12 | 0.6388779181281424 | 0.6019390796930701 | 0.036938838435072285 | 0.023900248151587384 |  | 0.15 | 0.6614132007004706 | 0.6244043973845335 | 0.037008803315937144 | 0.027986455422342755 |  | 0.18 | 0.684700509904958 | 0.6476075293752792 | 0.0370929805296788 | 0.03226617928811607 |  | 0.2 | 0.7087606089648042 | 0.671568760847692 | 0.03719184811711218 | 0.03674554440924512 |  | 0.23 | 0.7336147170961721 | 0.6963088256472445 | 0.0373058914489276 | 0.04143081866361794 |  | 0.25 | 0.7592845190779273 | 0.7218489158019433 | 0.03743560327598394 | 0.04632841624970555 |  | 0.28 | 0.7857921751780926 | 0.7482106913893737 | 0.03758148378871895 | 0.05144490090694335 |  | 0.3 | 0.8131603314505625 | 0.7754162907643443 | 0.03774404068621828 | 0.056786989257941047 |  | 0.33 | 0.8414121304155557 | 0.8034883411600917 | 0.037923789255464024 | 0.062361554276979946 |  | 0.35 | 0.8705712221372056 | 0.8324499696759273 | 0.03812125246127829 | 0.06817562888922864 |  | 0.38 | 0.9006617757115863 | 0.8623248146641316 | 0.038336961047454765 | 0.07423640970507546 |  | 0.4 | 0.9317084911783655 | 0.8931370375288183 | 0.03857145364954717 | 0.08055126089393987 |  | 0.43 | 0.9637366118691482 | 0.9249113349493694 | 0.03882527691977877 | 0.0871277182018827 |  | 0.45 | 0.9967719372054492 | 0.957672951540947 | 0.039098985664502206 | 0.09397349311729353 |  | 0.48 | 1.0308408359590917 | 0.9914476929644636 | 0.03939314299462815 | 0.10109647718888563 |  | 0.5 | 1.0659702599876857 | 1.0262619394982737 | 0.039708320489411975 | 0.10850474650018253 |  | 0.53 | 1.1021877584577024 | 1.0621426600837252 | 0.040045098373977295 | 0.11620656630463329 |  | 0.55 | 1.1395214925675101 | 1.0991174268565946 | 0.04040406571091548 | 0.1242103958254438 |  | 0.58 | 1.1780002507826035 | 1.1372144301763043 | 0.040785820606299206 | 0.1325248932241675 |  | 0.6 | 1.2176534645951194 | 1.1764624941647137 | 0.04119097043040565 | 0.14115892074205152 |  | 0.62 | 1.2585112248196084 | 1.216891092766168 | 0.041620132053440395 | 0.15012155001809477 |  | 0.65 | 1.3006042984369122 | 1.2585303663403975 | 0.042073932096514666 | 0.1594220675877326 |  | 0.68 | 1.3439641459978935 | 1.3014111387997649 | 0.0425530071981286 | 0.16906998056603173 |  | 0.7 | 1.3886229395986762 | 1.3455649353023105 | 0.04305800429636575 | 0.17907502251924487 |  | 0.73 | 1.4346135814389764 | 1.3910240005119654 | 0.04358958092701104 | 0.18944715952855415 |  | 0.75 | 1.4819697229750504 | 1.4378213174372818 | 0.044148405537768554 | 0.2001965964498101 |  | 0.78 | 1.5307257846787516 | 1.4859906268600118 | 0.044735157818739824 | 0.21133378337306466 |  | 0.8 | 1.5809169764141728 | 1.5355664473648554 | 0.04535052904931747 | 0.22286942228568918 |  | 0.83 | 1.6325793184433537 | 1.5865840959817388 | 0.045995222461614915 | 0.23481447394287258 |  | 0.85 | 1.685749663072577 | 1.6390797094520182 | 0.04666995362055881 | 0.24718016494930675 |  | 0.88 | 1.7404657169508255 | 1.6930902661300868 | 0.047375450820738685 | 0.2599779950558842 |  | 0.9 | 1.79676606403206 | 1.7486536085319457 | 0.048112455500114226 | 0.2732197446752623 |  | 0.93 | 1.854690189213094 | 1.805808466542438 | 0.048881722670655936 | 0.28691748262018724 |  | 0.95 | 1.9142785026589775 | 1.8645944812929833 | 0.04968402136599415 | 0.3010835740685152 |  | 0.98 | 1.9755723648279737 | 1.9250522297218278 | 0.05052013510614595 | 0.31573068875892857 |  | 1.0 | 2.0386141122084127 | 1.9872232498290403 | 0.051390862379372404 | 0.33087180942140754 |  +------+--------------------+--------------------+----------------------+----------------------+  Finite difference method  +------+--------------------+--------------------+------------------------+----------------------+  | X | Y | Accurate Y | |Y - Accurate Y| | Runge eps |  +------+--------------------+--------------------+------------------------+----------------------+  | 0 | 0.4999404725372249 | 0.5 | 5.952746277509968e-05 | 4.46430017150834e-05 |  | 0.03 | 0.5190029632360589 | 0.5190654460249184 | 6.24827888595636e-05 | 0.009564723278543186 |  | 0.05 | 0.5387082091752429 | 0.5387736998918138 | 6.54907165709151e-05 | 0.01965838647643059 |  | 0.08 | 0.5590743822383485 | 0.5591429350060964 | 6.855276774786478e-05 | 0.030252284345585134 |  | 0.1 | 0.5801200600545191 | 0.5801917305967077 | 7.167054218859548e-05 | 0.04136273550613734 |  | 0.12 | 0.6018642339727579 | 0.6019390796930701 | 7.484572031224435e-05 | 0.05300644718027081 |  | 0.15 | 0.6243263173187041 | 0.6244043973845335 | 7.808006582932236e-05 | 0.06520052320615921 |  | 0.17 | 0.6475261539468589 | 0.6476075293752792 | 8.13754284203494e-05 | 0.07796247234313924 |  | 0.2 | 0.6714840271012723 | 0.671568760847692 | 8.473374641970732e-05 | 0.09131021688103358 |  | 0.22 | 0.6962206685977359 | 0.6963088256472445 | 8.815704950859704e-05 | 0.10526210156656413 |  | 0.25 | 0.7217572683405281 | 0.7218489158019433 | 9.164746141521185e-05 | 0.11983690285980386 |  | 0.27 | 0.74811548418675 | 0.7482106913893735 | 9.520720262345961e-05 | 0.13505383853360242 |  | 0.3 | 0.7753174521712538 | 0.7754162907643443 | 9.883859309045562e-05 | 0.15093257762888712 |  | 0.33 | 0.8033857971051178 | 0.8034883411600917 | 0.00010254405497389651 | 0.1674932507786897 |  | 0.35 | 0.8323436435605518 | 0.8324499696759273 | 0.00010632611537553238 | 0.1847564609136827 |  | 0.38 | 0.8622146272550395 | 0.8623248146641316 | 0.00011018740909207736 | 0.2027432943619295 |  | 0.4 | 0.8930229068474291 | 0.8931370375288183 | 0.0001141306813892129 | 0.2214753323554558 |  | 0.43 | 0.9247931761585795 | 0.9249113349493695 | 0.00011815879079002389 | 0.24097466295615 |  | 0.45 | 0.9575506768290608 | 0.957672951540947 | 0.00012227471188619443 | 0.26126389341338907 |  | 0.48 | 0.9913212114262885 | 0.9914476929644638 | 0.00012648153817529373 | 0.2823661629656652 |  | 0.5 | 1.0261311570133498 | 1.0262619394982737 | 0.0001307824849239303 | 0.30430515609837305 |  | 0.53 | 1.0620074791916605 | 1.0621426600837254 | 0.00013518089206487893 | 0.3271051162697919 |  | 0.55 | 1.0989777466294681 | 1.0991174268565949 | 0.00013968022712673722 | 0.3507908601171791 |  | 0.58 | 1.1370701460880979 | 1.1372144301763045 | 0.00014428408820665872 | 0.37538779215476914 |  | 0.6 | 1.1763134979577288 | 1.176462494164714 | 0.00014899620698516358 | 0.40092191997536486 |  | 0.63 | 1.2167372723143786 | 1.2168910927661685 | 0.00015382045178991 | 0.4274198699670986 |  | 0.65 | 1.258371605509685 | 1.2585303663403975 | 0.00015876083071253433 | 0.45490890355685254 |  | 0.68 | 1.3012473173049801 | 1.301411138799765 | 0.0001638214947849992 | 0.48341693399173224 |  | 0.7 | 1.3453959285610937 | 1.345564935302311 | 0.0001690067412172258 | 0.5129725436699317 |  | 0.73 | 1.3908496794952594 | 1.3910240005119658 | 0.00017432101670644684 | 0.543605002032264 |  | 0.75 | 1.437641548516465 | 1.4378213174372823 | 0.00017976892081716933 | 0.5753442840256001 |  | 0.78 | 1.4858052716505679 | 1.485990626860012 | 0.00018535520944418238 | 0.6082210891494386 |  | 0.8 | 1.5353753625664972 | 1.5355664473648558 | 0.00019108479835860948 | 0.6422668610968303 |  | 0.83 | 1.586387133214894 | 1.586584095981739 | 0.00019696276684499914 | 0.6775138080009104 |  | 0.85 | 1.6388767150905814 | 1.639079709452019 | 0.00020299436143766947 | 0.7139949232983357 |  | 0.88 | 1.6928810811303312 | 1.6930902661300877 | 0.0002091849997565287 | 0.7517440072209931 |  | 0.9 | 1.7484380682574912 | 1.7486536085319466 | 0.00021554027445547241 | 0.7907956889274517 |  | 0.93 | 1.8055864005851558 | 1.8058084665424394 | 0.00022206595728357925 | 0.8311854492857422 |  | 0.95 | 1.8643657132897182 | 1.8645944812929844 | 0.00022876800326621094 | 0.8729496443192091 |  | 0.98 | 1.924816577166812 | 1.9250522297218287 | 0.00023565255501667437 | 0.9161255293273474 |  | 1.0 | 1.9869805238818614 | 1.9872232498290414 | 0.0002427259471800003 | 0.96075128369375 |  +------+--------------------+--------------------+------------------------+----------------------+ |

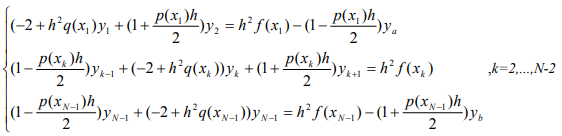
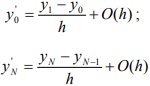
Выводы.

Выполнив лабораторную работу, я научился реализовывать методы решения задач Коши и краевых задач: метод Эйлера (*O(h)*), метод Рунге-Кутты четвёртого порядка, метод Адамса четвёртого порядка, метод стрельбы, метод конечных разностей (*O(h)* или *O(h/2)*). Существует несколько модифицированных методов Эйлера со вторым порядком точности.

Для решения жёстких систем ОДУ существуют неявные методы, например неявный метод Эйлера, использующий производную на правой границе: .

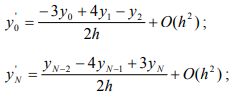


В методе Адамса можно использовать односторонние разности первого порядка, тогда мы получаем трёхдиагональную матрицу коэффициентов .



Или односторонние разности второго порядка,

, тогда второй порядок аппроксимации сохраняется везде, но матрица линейной системы не трехдиагональная. Первый подход лучше использовать, когда требуется высокая скорость вычислений, второй – когда более важна высокая точность решения.



Исходный код.

import argparse

import numpy as np

from prettytable import PrettyTable

def foo(x, y, y1):

return 2 \* y1 \* np.tan(x) - 3 \* y

def orig\_foo(x):

#return (np.cos(x))\*\*3 + np.sin(x) \* (1 + 2 \* (np.cos(x))\*\*2)

return (np.cos(2\*x) + np.sin(2\*x)) / ((np.cos(x)))

def euler(f, xa, xb, ya, y1a, h):

n = int((xb - xa) / h)

x = xa

y = ya

x\_res = [x]

y\_res = [y]

y1 = y1a

for i in range(n):

y1 += h \* f(x, y, y1)

y += h \* y1

x += h

x\_res.append(x)

y\_res.append(y)

return x\_res, y\_res

def runge\_kutta(f, xa, xb, ya, y1a, h):

n = int((xb - xa) / h)

x = xa

y = ya

z = y1a

x\_res = [x]

y\_res = [y]

z\_res = [z]

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

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 adams(f, x, y, h, n, z):

z = z[:4] + [0] \* (len(z) - 4)

for i in range(3, n):

z[i + 1] = z[i] + h/24 \* (55\*f(x[i], y[i], z[i]) - \

59\*f(x[i - 1], y[i - 1], z[i - 1]) + \

37\*f(x[i - 2], y[i - 2], z[i - 2]) - \

9\*f(x[i - 3], y[i - 3], z[i - 3]))

tmp = y[i] + h/24 \* (55\*z[i] - 59\*z[i - 1] + \

37\*z[i - 2] - 9\*z[i - 3])

x.append(x[-1] + h)

y.append(tmp)

return x, y

def print\_result\_table(f, name, res, runge\_y, p):

f.write(f'\n{name}\n')

table = PrettyTable(['X', 'Y', 'Accurate Y', 'Runge eps'])

for x, y, yr in zip(\*res, runge\_y):

tmp = orig\_foo(x)

table.add\_row([round(x, 2), round(y, 3), round(tmp, 3),

abs(y - yr) / (2\*\*p - 1)])

f.write(f'\n{str(table)}\n')

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--output', required=True, help='File for answer')

parser.add\_argument('--h', required=True, help='Step', type=float)

args = parser.parse\_args()

a = 0

b = 1

y0 = 1

y10 = 2

step = args.h

f = open(args.output, 'w')

res1 = euler(foo, a, b, y0, y10, step)

res1\_half\_h = euler(foo, a, b, y0, y10, step / 2)

print\_result\_table(f, 'Euler', res1, res1\_half\_h[1], 1)

res2, z = runge\_kutta(foo, a, b, y0, y10, step)

res2\_half\_h, z\_half\_h = runge\_kutta(foo, a, b, y0, y10, step / 2)

print\_result\_table(f, 'Runge-Kutta', res2, res2\_half\_h[1], 4)

res3 = adams(foo, res2[0][:4], res2[1][:4], step, int((b - a) / step), z)

res3\_half\_h = adams(foo, res2[0][:4], res2[1][:4], step / 2,

int((b - a) / step), z\_half\_h)

print\_result\_table(f, 'Adams', res3, res3\_half\_h[1], 4)

f.close()

if \_\_name\_\_ == "\_\_main\_\_":

main()

import argparse

import numpy as np

from prettytable import PrettyTable

def func(x, y, y1):

return (2\*y1 + np.exp(x)\*y)/ (np.exp(x) + 1)

def orig\_func(x):

return np.exp(x) - 1 + 1/(np.exp(x)+1)

def p(x):

return -2/(np.exp(x)+1)

def q(x):

return -np.exp(x)/(np.exp(x)+1)

def f(x):

return 0

def tma(a, b, c, d, shape):

p = [-c[0] / b[0]]

q = [d[0] / b[0]]

x = [0] \* (shape + 1)

for i in range(1, shape):

p.append(-c[i] / (b[i] + a[i] \* p[i - 1]))

q.append((d[i] - a[i] \* q[i - 1]) / (b[i] + a[i] \* p[i - 1]))

for i in reversed(range(shape)):

x[i] = p[i] \* x[i + 1] + q[i]

return x[:-1]

def runge\_kutta(f, xa, xb, ya, y1a, h):

n = int((xb - xa) / h)

x = xa

y = ya

z = y1a

x\_res = [x]

y\_res = [y]

z\_res = [z]

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

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):

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

return cur\_n - num4 \* (cur\_n - prev\_n) / num5

def shooting\_method(a, b, h, eps, f, alpha0, alpha1, beta0, beta1, A, B):

n\_prev = 1

n\_cur = 0.8

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]

while abs(alpha1 \* ans\_cur[1][len(ans\_cur[0]) - 1] + \

beta1 \* der\_one(ans\_cur[0], ans\_cur[1], b) - B) > eps:

n = next\_n(n\_cur, n\_prev, ans\_cur, ans\_prev, alpha1, beta1, B, b)

n\_prev = n\_cur

n\_cur = n

ans\_prev = ans\_cur

ans\_cur = runge\_kutta(f, a, b, n\_cur, (A - alpha0\*n\_cur) / beta0, h)[0]

return ans\_cur

def finite\_difference\_method(a1, b1, h, alpha\_0, alpha\_1, beta\_0, beta\_1, A, B):

x = [a1]

a = []

b = []

c = []

d = []

n = round((b1 - a1) / h)

a.append(0)

b.append(-2 / (h \* (2 - p(a1) \* h)) + q(a1) \* h /

(2 - p(a1) \* h) + alpha\_0 / beta\_0)

c.append(2 / (h \* (2 - p(a1) \* h)))

d.append(A / beta\_0 + h \* f(a1) / (2 - p(a1) \* h))

x.append(x[0] + h)

for i in range(1, n):

a.append(1 / h\*\*2 - p(x[i]) / (2 \* h))

b.append(-2 / h\*\*2 + q(x[i]))

c.append(1 / h\*\*2 + p(x[i]) / (2 \* h))

d.append(f(x[i]))

x.append(x[i] + h)

a.append(-2 / (h \* (2 + p(x[n]) \* h)))

b.append(2 / (h \* (2 + p(x[n]) \* h)) - q(x[n]) \* h /

(2 + p(x[n]) \* h) + alpha\_1 / beta\_1)

c.append(0)

d.append(B / beta\_1 - h \* f(x[n]) / (2 + p(x[n]) \* h))

y = tma(a, b, c, d, len(a))

return x, y

def print\_result\_table(f, name, res, runge\_y, p):

f.write(f'\n{name}\n')

table = PrettyTable(['X', 'Y', 'Accurate Y', '|Y - Accurate Y|',

'Runge eps'])

for x, y, yr in zip(\*res, runge\_y):

tmp = orig\_func(x)

table.add\_row([round(x, 2), y, tmp, abs(y - tmp),

abs(y - yr) / (2\*\*p - 1)])

f.write(f'\n{str(table)}\n')

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--output', required=True, help='File for answer')

parser.add\_argument('--h', required=True, help='Step', type=float)

parser.add\_argument('--eps', type=float, help='Epsilon', default=1e-5)

args = parser.parse\_args()

a = 0

b = 1

alpha0 = 0

alpha1 = 0

beta0 = 1

beta1 = 1

y0 = 3/4

y10 = (np.e\*\*2)\*(np.e+2)/(np.e+1)\*\*2

step = args.h

eps = args.eps

f = open(args.output, 'w')

res1 = shooting\_method(a, b, step, eps, func, alpha0, alpha1,

beta0, beta1, y0, y10)

res2 = shooting\_method(a, b, step / 2, eps, func, alpha0, alpha1,

beta0, beta1, y0, y10)

print\_result\_table(f, 'Shooting method', res1, res2[1], 2)

res1 = finite\_difference\_method(a, b, step, alpha0, alpha1,

beta0, beta1, y0, y10)

res2 = finite\_difference\_method(a, b, step / 2, alpha0, alpha1,

beta0, beta1, y0, y10)

print\_result\_table(f, 'Finite difference method', res1, res2[1], 1)

f.close()

if \_\_name\_\_ == "\_\_main\_\_":

main()