Дисциплина: Численные методы

Лабораторное задание №3

Отчет

Тема: «Решение краевой задачи ОДУ методом пристрелки»

Выполнил:

студент 3 курса 8 группы

Крутько А.С.

Проверила:

преподаватель

Махинова О.А.

[Задача 1 3](#_Toc101009838)

[Постановка задачи 3](#_Toc101009839)

[Используемые формулы 4](#_Toc101009840)

[Реализация формул в коде 6](#_Toc101009841)

[Численные эксперименты 8](#_Toc101009842)

[Задача 2 9](#_Toc101009843)

[Постановка задачи 9](#_Toc101009844)

[Используемые формулы 10](#_Toc101009845)

[Реализация формул в коде 12](#_Toc101009846)

[Численные эксперименты 14](#_Toc101009847)

[Вывод 16](#_Toc101009848)

Задача 1

Постановка задачи

Решить краевую задачу для обыкновенного дифференциального уравнения второго порядка методом пристрелки.

Решается обыкновенное дифференциальное уравнение:

С граничными условиями типа:

И направлением интегрирования

Для решения задачи Коши используется любой из методов Рунге-Кутта. Для решения возникающего нелинейного уравнения – метод половинного деления.

Решение нелинейного уравнения , из которого определяется значение пристрелочного параметра вести до тех пор, пока

Используемые формулы

При выполнении данной задачи лабораторной работы №3. Мною были использованы следующие формулы и методы решения задач:

1. Метод Рунге-Кутта второго порядка для решения задачи Коши
2. Метод половинного деления для решения возникающего нелинейного уравнения

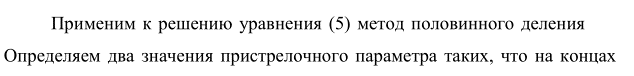
***Метод Рунге-Кутта второго порядка***

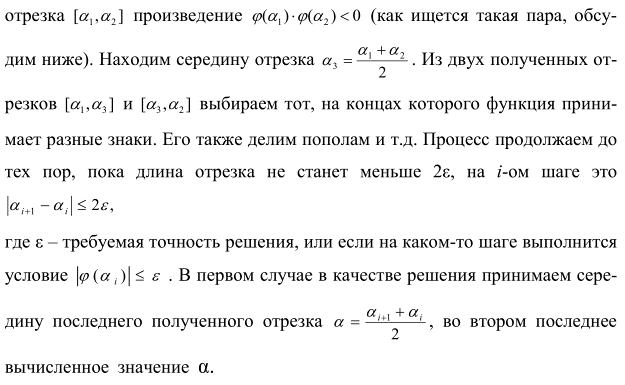
Данный метод был использован в следующем виде:

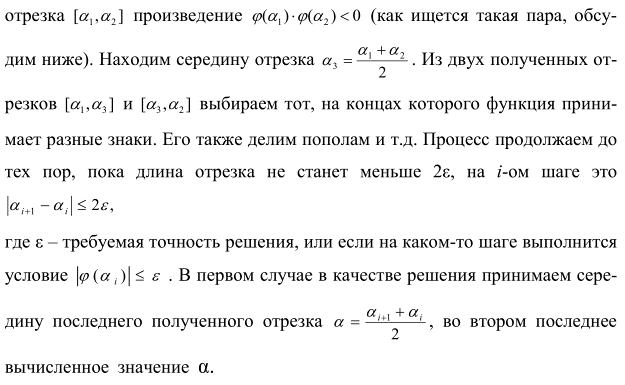
Где и – обозначения, введенные при привидении задачи (1), (2), (3) к системе двух дифференциальных уравнений первого порядка:

***Метод половинного деления***

Для решения возникающей при данном привидении задачи (1), (2), (3) к виду (4) нелинейной задачи использовался метод половинного деления. Описать его можно следующим образом:

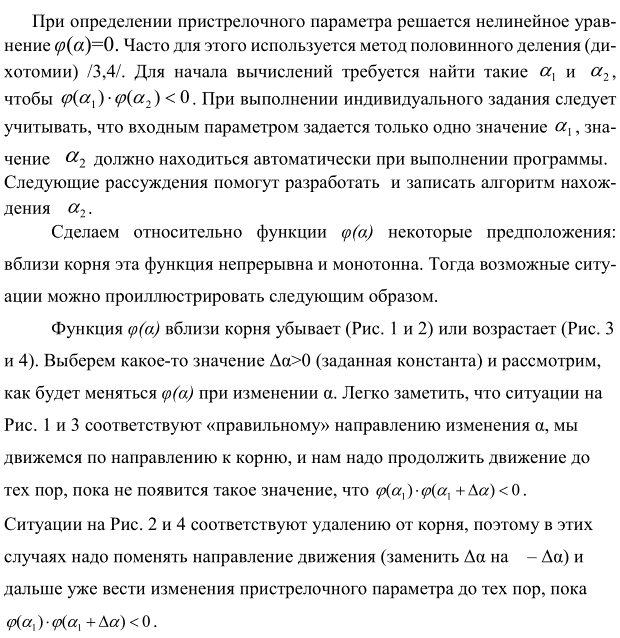


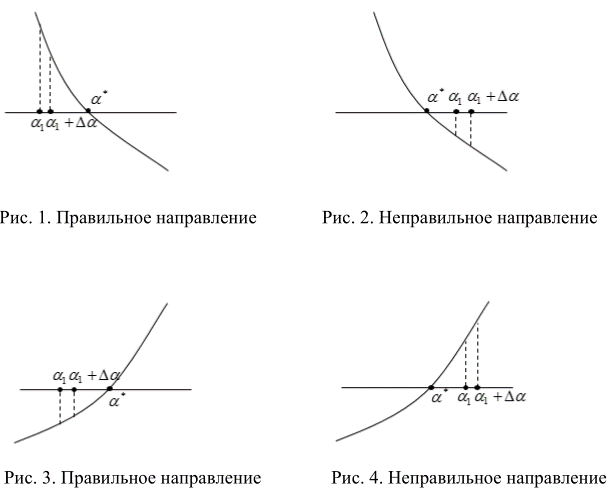




После, используя полученное значение – пристрелочного параметра, решаем получающуюся задачу Коши. Это решение и будет решением краевой задачи (4) в пределах заданной точности.

Пара параметров ищется с помощью следующего алгоритма:





Реализация формул в коде

***Формула для поиска :***

def find\_alpha2\_value(alpha1):  
 delta\_alpha = 1  
 alpha2 = alpha1 + delta\_alpha  
 fi\_alpha1 = fi\_alpha(alpha1)  
 fi\_alpha2 = fi\_alpha(alpha2)  
 flag\_reversed = False  
 while not different\_sign(fi\_alpha1, fi\_alpha2):  
 alpha2 = alpha1 + delta\_alpha  
 fi\_alpha2 = fi\_alpha(alpha2)  
 if fi\_alpha1 < 0:  
 if fi\_alpha2 > fi\_alpha1 and (not flag\_reversed):  
 delta\_alpha += 0.1  
 elif fi\_alpha2 > fi\_alpha1 and flag\_reversed:  
 delta\_alpha -= 0.1  
 elif fi\_alpha2 < fi\_alpha1:  
 delta\_alpha = delta\_alpha \* -1  
 flag\_reversed = True  
 if fi\_alpha1 > 0:  
 if fi\_alpha2 > fi\_alpha1:  
 delta\_alpha = delta\_alpha \* -1  
 flag\_reversed = True  
 else:  
 if fi\_alpha2 < fi\_alpha1 and (not flag\_reversed):  
 delta\_alpha += .2  
 if fi\_alpha2 < fi\_alpha1 and flag\_reversed:  
 delta\_alpha -= .2  
 return alpha2

***Метод половинного деления:***

# МПД  
def bisection(alpha1, alpha2, epsilon):  
 alpha3 = (alpha1 + alpha2) / 2  
 fi\_alpha1 = fi\_alpha(alpha1)  
 fi\_alpha2 = fi\_alpha(alpha2)  
 fi\_alpha3 = fi\_alpha(alpha3)  
 iterations\_count = 1  
 while math.fabs(fi\_alpha1 - fi\_alpha2) > epsilon and iterations\_count < K:  
 if math.fabs(fi\_alpha2) <= epsilon:  
 # Возвращаем значение альфа  
 print('Завершение работы внутри цикла')  
 return alpha2, iterations\_count, 0  
 if different\_sign(fi\_alpha1, fi\_alpha3):  
 alpha2 = alpha3  
 elif different\_sign(fi\_alpha3, fi\_alpha2):  
 alpha1 = alpha3  
 alpha3 = (alpha1 + alpha2) / 2  
 fi\_alpha1 = fi\_alpha(alpha1)  
 fi\_alpha2 = fi\_alpha(alpha2)  
 fi\_alpha3 = fi\_alpha(alpha3)  
 iterations\_count = iterations\_count + 1  
 if iterations\_count >= K:  
 return alpha3, iterations\_count, 1  
 print('Завершение работы вне цикла')  
 return alpha3, iterations\_count, 0

***Метод Рунге-Кутта:***

# Метод Рунге Кутта 2 порядка:  
def RungeKut(input\_alpha):  
 input\_u = np.zeros(N + 1)  
 input\_w = np.zeros(N + 1)  
 input\_u[0] = A # u  
 input\_w[0] = input\_alpha # w  
 k1 = np.zeros(2)  
 k2 = np.zeros(2)  
  
 for i in range(N):  
 k1[0] = h \* du\_dx(x\_array[i], input\_u[i], input\_w[i])  
 k1[1] = h \* dw\_dx(x\_array[i], input\_u[i], input\_w[i])  
  
 k2[0] = h \* du\_dx(x\_array[i] + h, input\_u[i] + k1[0], input\_w[i] + k1[1])  
 k2[1] = h \* dw\_dx(x\_array[i] + h, input\_u[i] + k1[0], input\_w[i] + k1[1])  
  
 input\_u[i + 1] = input\_u[i] + 0.5 \* (k1[0] + k2[0])  
 input\_w[i + 1] = input\_w[i] + 0.5 \* (k1[1] + k2[1])  
 return input\_u, input\_w

***Метод для решения нелинейного уравнения***

def fi\_alpha(input\_alpha):  
 tmp\_u, \_ = RungeKut(input\_alpha)  
 fi\_alpha\_value = tmp\_u[N] - B  
 return fi\_alpha\_value

Численные эксперименты

Мною был проведен численный эксперимент со следующей задачей:

В результате преобразований данная задача приводится к следующей задаче Коши:

Где – пристрелочный параметр, который нужно получить в результате работы программы.

В результате численного эксперимента для следующих значений:

* epsilon = 1e-13;
* alpha = 0;
* A = 1;
* B = 8;

Был получен следующий результат:

Фи в альфа = 4.263256414560601e-14

Количество итераций 47

Конечное значение параметра пристрелки 2.0000000000003

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | y | deltaU |  | deltaU’ |
| 1.0 | 1.0 | 0.0 | 2.0 | 9.992007221626409e-14 |
| 1.002 | 1.004004 | 0.0 | 2.004 | 2.3904386914352926e-08 |
| 1.004 | 1.008016 | 4.771338879550058e-11 | 2.008 | 4.7618675669269805e-08 |
| … | … | … | … | … |
| 1.996 | 3.984016 | 2.872164766021257e-06 | 3.992 | 4.632178671748477e-06 |
| 1.998 | 3.992004 | 2.881428491807725e-06 | 3.996 | 4.637569378296291e-06 |
| 2.0 | 4.0 | 2.8907030076297247e-06 | 4.0 | 4.6429570756956196e-06 |

Задача 2

Постановка задачи

Решить краевую задачу для обыкновенного дифференциального уравнения второго порядка методом пристрелки.

Решается обыкновенное дифференциальное уравнение:

С граничными условиями типа:

Для решения задачи Коши используется любой из методов Рунге-Кутта. Для решения возникающего нелинейного уравнения – метод секущих.

Решение нелинейного уравнения , из которого определяется значение пристрелочного параметра вести до тех пор, пока

Используемые формулы

При выполнении данной задачи лабораторной работы №3. Мною были использованы следующие формулы и методы решения задач:

1. Метод Рунге-Кутта четвертого порядка для решения задачи Коши
2. Метод половинного деления для решения возникающего нелинейного уравнения

***Метод Рунге-Кутта четвертого порядка***

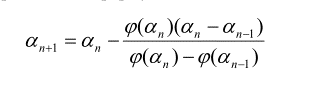
Данный метод был использован в следующем виде:

Где , и – обозначения, введенные при привидении задачи (1), (2), (3) к системе двух дифференциальных уравнений первого порядка:

***Метод секущих***

Для решения возникающей при данном привидении задачи (1), (2), (3) к виду (4) нелинейной задачи использовался метод секущих. Данный метод, аналогично методу половинного деления, итерационным способом получает решение уравнения (5):

Пусть – заданы, пока значение функции удовлетворяет условию получим значения следующего параметра через формулу:



После, используя полученное значение – пристрелочного параметра, решаем получающуюся задачу Коши. Это решение и будет решением краевой задачи (4) в пределах заданной точности.

Реализация формул в коде

***Метод Рунге-Кутта четвёртого порядка***

def RungeKut(input\_alpha, f1=u\_function, f2=v\_function, f3=w\_function):  
 input\_u = np.zeros(N + 1)  
 input\_v = np.zeros(N + 1)  
 input\_w = np.zeros(N + 1)  
 input\_u[N] = B # u  
 input\_v[N] = input\_alpha # v  
 input\_w[N] = C # w  
 k1 = np.zeros(3)  
 k2 = np.zeros(3)  
 k3 = np.zeros(3)  
 k4 = np.zeros(3)  
  
 for i in reversed(range(0, N, 1)):  
 # k1  
 x\_value = x\_array[i + 1]  
 uvw\_values = [  
 input\_u[i + 1],  
 input\_v[i + 1],  
 input\_w[i + 1]  
 ]  
 uvw\_part = uvw\_values  
 k1[0] = - h \* f1(x\_value, uvw\_part[0], uvw\_part[1], uvw\_part[2])  
 k1[1] = - h \* f2(x\_value, uvw\_part[0], uvw\_part[1], uvw\_part[2])  
 k1[2] = - h \* f3(x\_value, uvw\_part[0], uvw\_part[1], uvw\_part[2])  
 # k2  
 uvw\_part = [  
 uvw\_values[0] + k1[0] \* 0.25,  
 uvw\_values[1] + k1[1] \* 0.25,  
 uvw\_values[2] + k1[2] \* 0.25  
 ]  
 k2[0] = - h \* f1(x\_value + h \* 0.25, uvw\_part[0], uvw\_part[1], uvw\_part[2])  
 k2[1] = - h \* f2(x\_value + h \* 0.25, uvw\_part[0], uvw\_part[1], uvw\_part[2])  
 k2[2] = - h \* f3(x\_value + h \* 0.25, uvw\_part[0], uvw\_part[1], uvw\_part[2])  
 # k3  
 uvw\_part = [  
 uvw\_values[0] + k2[0] \* 0.5,  
 uvw\_values[1] + k2[1] \* 0.5,  
 uvw\_values[2] + k2[2] \* 0.5  
 ]  
 k3[0] = - h \* f1(x\_value \* 0.5, uvw\_part[0], uvw\_part[1], uvw\_part[2])  
 k3[1] = - h \* f2(x\_value \* 0.5, uvw\_part[0], uvw\_part[1], uvw\_part[2])  
 k3[2] = - h \* f3(x\_value \* 0.5, uvw\_part[0], uvw\_part[1], uvw\_part[2])  
 # k4  
 uvw\_part = [  
 uvw\_values[0] + k1[0] - 2 \* k2[0] + k3[0],  
 uvw\_values[1] + k1[1] - 2 \* k2[1] + k3[1],  
 uvw\_values[2] + k1[2] - 2 \* k2[2] + k3[2]  
 ]  
 k4[0] = - h \* f1(x\_value + h, uvw\_part[0], uvw\_part[1], uvw\_part[2])  
 k4[1] = - h \* f2(x\_value + h, uvw\_part[0], uvw\_part[1], uvw\_part[2])  
 k4[2] = - h \* f3(x\_value + h, uvw\_part[0], uvw\_part[1], uvw\_part[2])  
  
 k = 1/6  
  
 input\_u[i] = uvw\_values[0] + k \* (k1[0] + 4 \* k3[0] + k4[0])  
 input\_v[i] = uvw\_values[1] + k \* (k1[1] + 4 \* k3[1] + k4[1])  
 input\_w[i] = uvw\_values[2] + k \* (k1[2] + 4 \* k3[2] + k4[2])  
  
 return input\_u, input\_v, input\_w

***Итоговый метод пристрелки***

def shooting\_mehod():  
 param\_alpha0 = alpha0  
 param\_alpha1 = alpha1  
  
 iterations\_count = 0  
 fi\_alpha0 = fi\_alpha(param\_alpha0)  
 fi\_alpha1 = fi\_alpha(param\_alpha1)  
  
 # Метод секущих для вычисления следующего приближения альфа  
 try:  
 param\_alpha2 = param\_alpha1 - fi\_alpha1 \* (param\_alpha1 - param\_alpha0) / (fi\_alpha1 - fi\_alpha0)  
 except ZeroDivisionError:  
 print('Деление на ноль перед циклом - 152')  
 return 2, None, iterations\_count  
  
 fi = fi\_alpha(param\_alpha2)  
 iterations\_count += 1  
  
 IER = 0  
 while m.fabs(fi) > epsilon and iterations\_count < K:  
 iterations\_count += 1  
 param\_alpha0 = param\_alpha1  
 param\_alpha1 = param\_alpha2  
 fi\_alpha0 = fi\_alpha1  
 fi\_alpha1 = fi  
 try:  
 param\_alpha2 = param\_alpha1 - fi\_alpha1 \* (param\_alpha1 - param\_alpha0) / (fi\_alpha1 - fi\_alpha0)  
 except ZeroDivisionError:  
 print('Деление на ноль внутри цикла - 168')  
 return 2, None, iterations\_count  
 fi = fi\_alpha(param\_alpha2)  
 # Проверки на выход из итераций  
 if iterations\_count >= K:  
 # Код выхода "количество итераций превысило максимум"  
 IER = 1  
 return IER, param\_alpha2, iterations\_count  
 if m.fabs(fi) <= epsilon:  
 # Код выхода "все вычисления прошли успешно"  
 IER = 0  
 return IER, param\_alpha2, iterations\_count  
 return IER, param\_alpha2, iterations\_count

Численные эксперименты

Мною был проведен численный эксперимент со следующей задачей:

В результате вычислений данная задача приводится к виду:

Где – пристрелочный параметр, который нужно получить в результате работы программы.

В результате численного эксперимента для следующих значений:

* epsilon = 1e-14;
* alpha = 0;
* A = 0;
* B = 1;
* C = 6

Был получен следующий результат:

Количество итераций 2

Фи в альфа = -1.7763568394002505e-15

Конечное значение параметра пристрелки 2.99999999999999

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | y | deltaU | y’ | deltaU’ | y’’ | deltaU’’ |
| 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.1 | 0.001 | 0.0 | 0.03 | 0.0 | 0.6 | 0.0 |
| 0.2 | 0.008 | 1.999962695453661e-14 | 0.12 | 1.9997892231060632e-14 | 1.2 | 0.0 |
| 0.3 | 0.027 | 2.0001361678012586e-14 | 0.27 | 9.992007221626409e-15 | 1.8 | 9.992007221626409e-15 |
| 0.4 | 0.064 | 0.0 | 0.48 | 2.9976021664879227e-14 | 2.4 | 1.021405182655144e-14 |
| 0.5 | 0.125 | 3.9995784462121264e-14 | 0.75 | 2.9976021664879227e-14 | 3.0 | 3.9968028886505635e-14 |
| 0.6 | 0.216 | 6.000755448098971e-14 | 1.08 | 1.9984014443252818e-14 | 3.6 | 3.019806626980426e-14 |
| 0.7 | 0.343 | 6.000755448098971e-14 | 1.47 | 4.9960036108132044e-14 | 4.2 | 2.042810365310288e-14 |
| 0.8 | 0.512 | 3.9968028886505635e-14 | 1.92 | 3.9968028886505635e-14 | 4.8 | 6.039613253960852e-14 |
| 0.9 | 0.729 | 2.9976021664879227e-14 | 2.43 | 3.9968028886505635e-14 | 5.4 | 7.993605777301127e-14 |
| 1.0 | 1.0 | 4.9960036108132044e-14 | 3.0 | 9.015010959956271e-14 | 6.0 | 3.493605777301127e-14 |

Вывод

В результате выполнения лабораторной работы у меня получилось найти численные решения для краевых задач ОДУ второго и третьего порядков с помощью метода пристрелки, работа данного метода заключалась в следующем алгоритме:

1. Приведение изначальной задачи к системе дифференциальных уравнений
2. Добавление пристрелочного параметра
3. Решение возникающей при приведении задачи ОДУ к системе дифференциальных уравнений нелинейной задачи с помощью одного из методов оптимизации.
4. Использование полученного в предыдущем пункте решения для получения решения системы одним из методов семейства методов Рунге-Кутта.

Алгоритм метода пристрелки возможно использовать и для краевых задач, не имеющих аналитического решения.