|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

**Лабораторная работа №1**

**по курсу “Моделирование”**

**по теме “Приближённый аналитический метод Пикара в сравнении с численными методами”**

|  |  |
| --- | --- |
| Студент: | Уласик Е.А. |
| Группа: | ИУ7-61 |
| Преподаватель: | Градов В.М. |

*2020 г.***1. Введение**

Цель: провести сравнение аналитического метода Пикара с численными методами.

**2. Описание методов**

*2.1 Метод Пикара*

Формула (1) описывает общий вид поиска приближённой функции на итерации s по методу Пикара:

|  |  |
| --- | --- |
|  | (1) |

где – заданная функция. При нулевой итерации .

*2.2 Численные методы*

Формула (2) описывает явный численный метод:

|  |  |
| --- | --- |
|  | (2) |

где h – шаг

Формула (3) описывает неявный численный метод

|  |  |
| --- | --- |
|  | (3) |

**3. Постановка задачи**

Необходимо найти приближённое решение уравнения, которое не имеет аналитического решения. Система уравнений показана на Формуле (4).

|  |  |
| --- | --- |
|  | (4) |

Для примера найдём 4-ое приближение:

**4. Листинг кода**

Ниже дан листинг программы, написанной на языке Python, которая решает поставленную задачу.

import copy  
from math import sqrt  
  
  
class Polynomial:  
 def \_\_init\_\_(self, c: list, power: list):  
 self.coefficient = []  
 self.power = []  
  
 self.constant = 0  
 self.coefficient.extend(c)  
 self.power.extend(power)  
 if len(self.power) != len(self.coefficient):  
 print("Error in polynomial")  
 exit(1)  
  
 self.length = len(self.power)  
  
 def \_\_pow\_\_(self, n):  
 return self \* self  
  
 def \_\_mul\_\_(self, other):  
 if type(other) is Polynomial:  
 coefficient = [0 for \_ in range(self.length \* self.length)]  
 power = [0 for \_ in range(self.length \* self.length)]  
 i = 0  
 for j in range(self.length):  
 for k in range(self.length):  
 power[i] = self.power[j] + other.power[k]  
 coefficient[i] = self.coefficient[j] \* other.coefficient[k]  
 i += 1  
 self.coefficient = coefficient  
 self.power = power  
 self.length = len(coefficient)  
 return self  
  
 def \_\_add\_\_(self, other):  
 if type(other) is Polynomial:  
 self.coefficient.extend(other.coefficient)  
 self.power.extend(other.power)  
 self.length += other.length  
 elif type(other) is int or type(other) is float:  
 self.constant += other  
  
 for i in range(self.length):  
 if self.coefficient[i] == 0:  
 self.coefficient.pop(i)  
 self.power.pop(i)  
 self.length -= 1  
 i -= 1  
 return self  
  
 def integrate(self, n=1):  
 for i in range(n):  
 for j in range(self.length):  
 self.power[j] += 1  
  
 for j in range(self.length):  
 self.coefficient[j] \*= self.power[j]  
 return self  
  
 def \_\_str\_\_(self):  
 res = ''  
 for i in range(self.length):  
 if self.coefficient[i] == 0:  
 continue  
 res += "(1 / " + str(self.coefficient[i]) + ")" + ' \* x^' + str(self.power[i])  
  
 if i + 1 < self.length and self.coefficient[i + 1] != 0:  
 res += ' + '  
 return res  
  
 def get\_value(self, x):  
 value = 0  
 for i in range(self.length):  
 value += x \*\* self.power[i] / self.coefficient[i]  
 return value  
  
  
def func(x, u):  
 return x \*\* 2 + u \*\* 2  
  
  
def euler\_method(x, h, y=0):  
 try:  
 y = y + h \* func(x, y)  
 return y  
 except (OverflowError, TypeError):  
 return None  
  
  
def implicit\_euler\_method(x, h, y=0):  
 try:  
 y = 1 / (2 \* h) - sqrt(1 / (4 \* h \*\* 2) - (x + h) \*\* 2 - y / h)  
 except:  
 y = None  
 return y  
  
  
def find\_polinom(n: list, xi=0, eta=0):  
 sum\_poly = Polynomial([0], [0])  
 nmax = max(n)  
 res = []  
 for s in range(1, nmax + 1):  
 last\_poly = copy.deepcopy(sum\_poly) \*\* 2  
 sum\_poly = Polynomial([1], [2]).integrate() + last\_poly.integrate()  
 if s in n:  
 res.append(sum\_poly)  
  
 return res  
  
  
def output(s):  
 if type(s) == float:  
 if s > 1000000:  
 return '{:.8e}'.format(s)  
 return '{:.8f}'.format(s)  
 elif type(s) == int:  
 return str(s)  
 elif s is None:  
 return "-----"  
 else:  
 return s  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 while True:  
 try:  
 n = list(map(int, input("Enter powers' of needed polynomials: (count < 6): ").split(' ')))  
 if len(n) < 0:  
 print("Wrong input. Try again.")  
 break  
 except:  
 pass  
  
 polies = find\_polinom(n)  
  
 x = 0 # int(input("Enter xi: ")) default value is 0  
 eta = 0 # default value is 0  
 x\_end = float(input("Enter x: "))  
 h = float(input("Enter step: "))  
 step\_in\_table = int(input("Enter step of rows in table: "))  
  
 title = "| X |"  
 for i in range(len(n)):  
 title += "{}-е приб. Пикар|".format(n[i])  
  
 title += " Явный Эйлера | Неявный Эйлера|"  
 print(title)  
  
 y\_polies = 0  
  
 eul\_y = eta  
 imp\_eul\_y = eta  
 current\_value = 0  
 while x < x\_end:  
 eul\_y = euler\_method(x, h, eul\_y)  
 imp\_eul\_y = implicit\_euler\_method(x, h, imp\_eul\_y)  
  
 if current\_value % step\_in\_table == 0:  
 row = '|{:^9.5f}|'.format(x)  
 for i in range(len(n)):  
 y\_polies = polies[i].get\_value(x)  
 row += "{:^15.8f}|".format(y\_polies)  
 row += "{:^15s}|{:^15s}|".format(output(eul\_y), output(imp\_eul\_y))  
 print(row)  
 current\_value = 0  
  
 current\_value += 1  
 x += h