|  |  |
| --- | --- |
| 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 **ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.03 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 1 |

**Дисциплина:** Моделирование

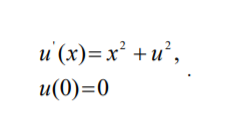
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-62Б |  |  | Н.А. Гарасев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | В.М. Градов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2021

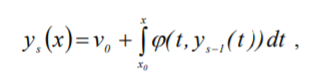
**Тема:** программная реализация приближенного аналитического метода и численных алгоритмов первого и второго порядков точности при решении задачи Коши для ОДУ.

**Цель работы:** получение навыков решения задачи Коши для ОДУ методами Пикара и явными методами первого порядка точности (Эйлера) и второго порядка точности (РунгеКутта).

**Входные данные:**



**Ход работы:**

Для метода Пикара необходимо вычислить многочлены приближения. Процедура последовательных приближений метода Пикара реализуется согласно следующей схеме: .

Данная процедура реализована автоматически в Листинге 1.

Листинг 1. Приближения Пикара.

**class** Polynom:  
 **def** \_\_init\_\_(self, k, d):  
 self.k = k  
 self.d = d  
  
 **def** integration(self):  
 self.d += 1  
 self.k /= self.d  
  
 **def** \_\_mul\_\_(self, other):  
 **if** isinstance(other, Polynom):  
 **return** Polynom(k=other.k \* self.k, d=other.d + self.d)  
 **else**:  
 **return** Polynom(k=other \* self.k, d=self.d)  
  
 **def** \_\_str\_\_(self):  
 **return f'k = {**self.k**}, d = {**self.d**} \n'  
  
 def** cop(self):  
 **return** self.k, self.d  
  
  
**def** opt(poly):  
 poly.sort(key=**lambda** x: x.d, reverse=**True**)  
 i = 0  
 **while** i < len(poly) - 1:  
 **if** poly[i].d == poly[i + 1].d:  
 poly[i].k += poly[i + 1].k  
 **del** poly[i + 1]  
 i -= 1  
 i += 1  
  
  
**def** integration(poly):  
 **for** i **in** poly:  
 i.integration()  
  
  
**def** f(poly):  
 tmp = []  
 **for** i **in** range(len(poly) - 1):  
 **for** j **in** range(i + 1, len(poly)):  
 tmp.append(poly[i] \* poly[j] \* 2)  
 **for** i **in** range(len(poly)):  
 poly[i].d \*= 2  
 poly[i].k = poly[i].k \*\* 2  
 tmp.append(poly[i])  
 **return** tmp  
  
  
**def** cop(poly):  
 tmp = []  
 **for** i **in** poly:  
 k, d = i.cop()  
 tmp.append(Polynom(k, d))  
 **return** tmp  
  
  
**def** init\_pickard():  
 pool = []  
 pol = [Polynom(1, 2)]  
 integration(pol)  
 pool.append(cop(pol))  
  
 **for** i **in** range(3):  
 pol = f(pol)  
 pol.append(Polynom(1, 2))  
 integration(pol)  
 opt(pol)  
 pool.append(cop(pol))  
 **return** pool  
  
  
**def** calc\_poly(pol, x, acc=3):  
 res = 0  
 **for** i **in** pol:  
 res += i.k \* (x \*\* i.d)  
 **return** round(res, acc)

Init\_pickar() – функция, с которой начинается метод. Первоначально создаем полином (x^2), от которого берем интеграл и запоминаем как метод Пикара с первым приближением. Затем прибавляем к нему (x^2) и повторяем предыдущие действия. Тем самым получаем метод Пикара с 4мя приближениями. Для вычисления значения приближения есть функция calc\_poly, которая принимает полином и точку x, в которой мы ищем значение. Остальные функции являются вспомогательными для оптимизации вычисления полиномов Пикара n-ого приближения.

Листинг 2. Метод Эйлера

**def** func(x, y):  
 **return** x \* x + y \* y  
  
  
**def** euler():  
 y = [0]  
 **for** i **in** range(1, len(x)):  
 y.append(y[i - 1] + step \* func(x[i - 1], y[i - 1]))  
 **return** y

Листинг 3. Метод Рунге-Кутта

**def** runge(a=0.5):  
 y = [0]  
 **for** i **in** range(1, len(x)):  
 k1 = func(x[i - 1], y[i - 1])  
 k2 = func(x[i - 1] + step / (2 \* a), y[i - 1] + step / (2 \* a) \* k1)  
 y.append(y[i - 1] + step \* ((1 - a) \* k1 + a \* k2))  
 **return** y

Листинг 4. Вывод результатов.

step = 10 \*\* -5  
x\_min = 0  
x\_max = 2.01  
skip = 100  
i = x\_min  
n = int((x\_max - x\_min) / step) + 1  
x = [x\_min + i \* step **for** i **in** range(n)]  
  
y\_euler = euler()  
y\_runge\_1 = runge(a=0.5)  
y\_runge\_2 = runge(a=1)

**for** i **in** range(n):  
 **if** i % skip:  
 **continue** tmp = **f'|{**round(i \* step, 3)**:<5}|'  
 for** pol **in** pool:  
 tmp += **f'{**output(calc\_poly(pol, i \* step))**}|'** tmp += **f'{**output(y\_euler[int(i)])**}|'** tmp += **f'{**output(y\_runge\_1[int(i)])**}|'** tmp += **f'{**output(y\_runge\_2[int(i)])**}|'** print(tmp)

Результат программы:

Программа выводит таблицу, содержащую значения аргумента с заданным шагом в интервале [0, xmax] и результаты расчета функции u(x) в приближениях Пикара (от 1-го до 4-го), а также численными методами. Границу интервала xmax выбирать максимально возможной из условия, чтобы численные методы обеспечивали точность вычисления решения уравнения u(x) до второго знака после запятой.