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**

О Т Ч Е Т

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:

$$u'(x) = x^2 + u^2,$$
$$u(0) = 0$$

Ход работы:

Для **метода Пикара** необходимо вычислить многочлены приближения. Процедура последовательных приближений **метода Пикара** реализуется согласно следующей схеме: $y_s(x) = v_0 + \int_{x_0}^{x} \varphi(t, y_{s-1}(t))\,dt$.

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.

Задачу Коши можно решить другими методами, такие как метод Эйлера и метод Рунге-Кутта.

Метод Эйлера.

$$y_{n+1} = y_n + hf(x_n, y_n)$$

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
```

Метод Рунге-Кутта

$$y_{n+1} = y_n + h[(1-\alpha)k_1 + \alpha k_2]\text{, где}$$

$$k_1 = f(x_n, y_n)$$

$$k_2 = f(x_n + \frac{h}{2\alpha}, y_n + \frac{h}{2\alpha}k_1) \quad , \quad \alpha = \frac{1}{2} \text{ или } 1$$

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.

Пример вывода.

X	Пикар 1	Пикар 2	Пикар 3	Пикар 4	Эйлер	Рунге a=0.5	Рунге a=1
0.0	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.002	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.003	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.004	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.005	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.006	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.007	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.008	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.009	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.01	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.011	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.012	0.000	0.000	0.000	0.000	0.000	0.000	0.000

1.996	2.651	4.654	7.116	10.262	138.949	139.902	139.902
1.997	2.655	4.665	7.141	10.317	161.338	162.663	162.663
1.998	2.659	4.676	7.167	10.372	192.315	194.267	194.267
1.999	2.663	4.687	7.193	10.428	237.991	241.111	241.111
2.0	2.667	4.698	7.219	10.484	312.061	317.721	317.720
2.001	2.671	4.710	7.245	10.540	452.860	465.680	465.678
2.002	2.675	4.721	7.271	10.597	823.630	871.532	871.516
2.003	2.679	4.732	7.298	10.654	4336.951	6771.104	6764.096
2.004	2.683	4.743	7.324	10.711	inf	inf	nan
2.005	2.687	4.754	7.351	10.769	inf	inf	nan

Вопросы.

1. Укажите интервалы значений аргумента, в которых можно считать решением заданного уравнения каждое из первых 4-х приближений Пикара. Точность результата оценивать до второй цифры после запятой. Объяснить свой ответ.

Каждое приближение Пикара вычисляет значение все точнее и точнее. А значит, что n-ое приближение вычисляет значения ближе к истинному, чем (n-1)-ое приближение.

Первое приближение: **от 0 до 0,665**

0.664	0.098	0.098
0.665	0.098	0.099
0.666	0.098	0.099
0.667	0.099	0.100
0.668	0.099	0.100

Второе приближение: **от 0 до 0,92**

0.919	0.259	0.268	0.268
0.92	0.260	0.268	0.269
0.921	0.260	0.269	0.270
0.922	0.261	0.270	0.271

Третье приближение**: от 0 до 1,184**

1.183	0.552	0.603	0.610	0.610
1.184	0.553	0.605	0.611	0.612
1.185	0.555	0.607	0.613	0.614
1.186	0.556	0.608	0.615	0.616

Для четвертого приближения Пикара нам необходимо знать пятое приближение Пикара, либо можно воспользоваться численными методами.

1.357	0.833	0.967	0.997	1.002	1.002	1.002	1.002
1.358	0.835	0.970	1.000	1.005	1.005	1.005	1.005
1.359	0.837	0.973	1.002	1.007	1.008	1.008	1.008
1.36	0.838	0.975	1.005	1.010	1.011	1.011	1.011
1.361	0.840	0.978	1.008	1.013	1.014	1.014	1.014
1.362	0.842	0.980	1.011	1.016	1.017	1.017	1.017

Четвертое приближение: **от 0 до 1,359**

Все вычисления производились для шага 10^-6

2. Пояснить, каким образом можно доказать правильность полученного результата при фиксированном значении аргумента в численных методах.

Численные методы зависят от шага. Возьмем точку x=2. Посмотрим, как меняются значения в этой точке в зависимости от численных методов.

Шаг = 10^-3

Эйлер: 126.597

Рунге-Кутта (a=0.5): 305.208

Рунге-Кутта (a=1): 300.663

Шаг = 10^-4

Эйлер: 270.068

Рунге-Кутта (a=0.5): 317.566

Рунге-Кутта (a=1): 317.490

Шаг = 10^-5

Эйлер: 312.061

Рунге-Кутта (a=0.5): 317.721

Рунге-Кутта (a=0.5): 317.720

Шаг = 10^-6

Эйлер: 317.145

Рунге-Кутта (a=0.5): 317.722

Рунге-Кутта (a=1): 317.722

Шаг = 10^-7

Эйлер: 317.665

Рунге-Кутта (a=0.5): 317.722

Рунге-Кутта (a=1): 317.722

Получается, что для метода Рунге-Кутта для значения x=2 достаточно шага 10^-5. Для метода Эйлера для значения x=2 при шаге = 10^-7 ответ с точностью до 2х знаков после запятой отличается от метода Рунге-Кутта. При попытке вычисления с шагом = 10^-8 программа выдает ошибку памяти.

Вывод:

При уменьшении шага точность увеличивается, однако компьютер работает с ограниченной разрядной сеткой, а значит нельзя знак бесконечно приближать к нулю.

3. **Каково значение функции при x = 2, т.е. привести значение u(2).**

При шаге = 10^-7

Эйлер: u(2) = 317.665

Рунге-Кутта (a=0.5): u(2) = 317.722

Рунге-Кутта (a=1): u(2) = 317.722