НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО
Факультет “Компьютерных технологий в дизайне”

Лабораторная работа № 6
по дисциплине “Вычислительная математика”
вариант №21

Выполнили:
Савельева Елизавета Юрьевна
Фурзикова Александра Евгеньевна
Группа: P3266

Преподаватель:
Машина Екатерина Алексеевна

г. Санкт-Петербург
2025

Цель лабораторной работы: Решить задачу Коши для обыкновенных дифференциальных уравнений численными методами

Формулы:

Листинг программы:

```
import warnings
import matplotlib.pyplot as plt
import numpy as np
import numexpr as ne


class IncorrectValueException(Exception):
    def __init__(self, message):
        self.message = message


EQUATIONS = {
    1: "(x-y)**2",
    2: "(x-y)**2+y**2",
    3: "y+(1+x)*y**2"
}

ANSWERS = {
    1: "1/(c*exp(2*x) + 1/2) + x - 1",
    2: "x - c - sqrt(c**2 - x + c)",
    3: "-1/(c - x - x**2/2)"
}

C = {
    1: "((y - x + 1)**2 - 1)/exp(-2*x)",
    2: "x - y - sqrt(y**2 - (x - y)**2)",
    3: "x + x**2/2 + 1/y"
}

METHODS = [
    'Одношаговый. Метод Эйлера',
    'Одношаговый. Метод Рунге-Кутта 4-го порядка',
    'Многошаговый. Метод Адамса'
]
```

```python
def calculateFunction(xi, yi, function_number):
    return float(ne.evaluate(EQUATIONS[function_number], local_dict={'x': xi, 'y': yi}))


def calculateC(x0, y0, function_number):
    try:
        return float(ne.evaluate(C[function_number], local_dict={'x': x0, 'y': y0}))
    except Exception as e:
        print(f"Ошибка вычисления константы C: {str(e)}")
        return float('nan')


def calculateAnswer(x0, y0, xi, function_number):
    try:
        c_val = calculateC(x0, y0, function_number)
        if np.isnan(c_val) or np.isinf(c_val):
            return float('nan')
        result = ne.evaluate(ANSWERS[function_number], local_dict={'x': xi, 'c': c_val})
        return float(result)
    except Exception as e:
        print(f"Ошибка вычисления аналитического решения: {str(e)}")
        return float('nan')


class Validator:
    @staticmethod
    def validateNumber(number: str):
        try:
            return float(number)
        except ValueError:
            raise IncorrectValueException('Необходимо ввести число.')

    @staticmethod
    def validateEpsilon(epsilon: str):
        try:
            epsilon = float(epsilon)
            if epsilon >= 0:
                return epsilon
            raise IncorrectValueException('Точность должна быть >= 0.')
        except ValueError:
            raise IncorrectValueException('Точность должна быть числом.')

    @staticmethod
    def validateFunctionNumber():
        try:
            number = int(input())
            if 1 <= number <= len(EQUATIONS):
                return number
```

```
            raise IncorrectValueException('Недопустимый номер функции.')
        except ValueError:
            raise IncorrectValueException('Введите целое число.')

    @staticmethod
    def validateMethodNumber():
        try:
            number = int(input())
            if 1 <= number <= len(METHODS):
                return number
            raise IncorrectValueException('Недопустимый номер метода.')
        except ValueError:
            raise IncorrectValueException('Введите целое число.')

    @staticmethod
    def validateBorders(border_left: float, border_right: float):
        if border_left < border_right:
            return True
        raise IncorrectValueException('Левая граница должна быть меньше правой.')

    @staticmethod
    def validateH():
        try:
            h = float(input())
            if h > 0:
                return h
            raise IncorrectValueException('Шаг должен быть > 0.')
        except ValueError:
            raise IncorrectValueException('Шаг должен быть числом.')


class DifferentialEquations:
    def __init__(self, x0, y0, xn, h, eps):
        self.__x0 = x0
        self.__y0 = y0
        self.__xn = xn
        self.__h = h
        self.__eps = eps
        self.__x_array = list(np.arange(x0, xn + 1e-10, h))

    def getY0(self): return self.__y0

    def getX0(self): return self.__x0

    def getXN(self): return self.__xn

    def getH(self): return self.__h

    def getEps(self): return self.__eps
```

```
    def getArrayX(self): return self.__x_array

    def ruleRunge(self, I_h, I_h2, k, eps):
        delta = abs(I_h2 - I_h) / (2 ** k - 1)
        print(f'Погрешность: {delta} {"<" if delta < eps else ">="} {eps}')
        return delta <= eps


class DifferentialEquationsMethods(DifferentialEquations):
    # =================== МЕТОД ЭЙЛЕРА ===================
    def methodEiler(self, function_number):
        results = self._calculateEilerMethod(function_number)
        self._printEilerTable(results)
        return results

    def _calculateEilerMethod(self, function_number):
        x_arr = self.getArrayX()
        n = len(x_arr)
        h = self.getH()
        x0_val = self.getX0()
        y0_val = self.getY0()

        iterations = [[0.0] * 5 for _ in range(n)]

        for i in range(n):
            xi = x_arr[i]
            iterations[i][0] = i
            iterations[i][1] = round(xi, 5)

            if i == 0:
                y = y0_val
            else:
                x_prev = x_arr[i - 1]
                y_prev = iterations[i - 1][2]
                f_prev = calculateFunction(x_prev, y_prev, function_number)
                y = y_prev + h * f_prev

            iterations[i][2] = round(y, 5)
            iterations[i][3] = round(calculateFunction(xi, y, function_number), 5)
            iterations[i][4] = round(calculateAnswer(x0_val, y0_val, xi, function_number),
5)

        return iterations

    def _printEilerTable(self, iterations):
        print('\n\t\tМетод Эйлера')
        print('i  |    x    |    y    |   f(x,y)   | Точное решение |')
        for row in iterations:
            exact = row[4] if not np.isnan(row[4]) else "N/A"
            print(f"{int(row[0]):2} | {row[1]:7.5f} | {row[2]:7.5f} | {row[3]:9.5f} |
```

```
{exact:14} |")

    # =================== МЕТОД РУНГЕ-КУТТА ===================
    def methodRungeCutta4(self, function_number):
        results = self._calculateRungeCutta4(function_number)
        self._printRungeCuttaTable(results)
        return results

    def _calculateRungeCutta4(self, function_number):
        x_arr = self.getArrayX()
        n = len(x_arr)
        h = self.getH()
        x0_val = self.getX0()
        y0_val = self.getY0()
        iterations = [[0.0] * 9 for _ in range(n)]

        for i in range(n):
            xi = x_arr[i]
            iterations[i][0] = i
            iterations[i][1] = round(xi, 5)

            if i == 0:
                y = y0_val
            else:
                y = iterations[i - 1][2] + iterations[i - 1][7]

            iterations[i][2] = round(y, 5)

            try:
                k1 = h * calculateFunction(xi, y, function_number)
                k2 = h * calculateFunction(xi + h / 2, y + k1 / 2, function_number)
                k3 = h * calculateFunction(xi + h / 2, y + k2 / 2, function_number)
                k4 = h * calculateFunction(xi + h, y + k3, function_number)
                delta_y = (k1 + 2 * k2 + 2 * k3 + k4) / 6
            except Exception as e:
                print(f"Ошибка расчета коэффициентов: {str(e)}")
                k1 = k2 = k3 = k4 = delta_y = float('nan')

            iterations[i][3] = round(k1, 5)
            iterations[i][4] = round(k2, 5)
            iterations[i][5] = round(k3, 5)
            iterations[i][6] = round(k4, 5)
            iterations[i][7] = round(delta_y, 5)
            iterations[i][8] = round(calculateAnswer(x0_val, y0_val, xi, function_number),
5)

        return iterations

    def _printRungeCuttaTable(self, iterations):
        print('\n\t\tМетод Рунге-Кутта 4-го порядка')
```

```
        print('i  |     x     |     y     |     k1      |      k2      |      k3      |      k4      |   delta   | Точное решение
|')
        for row in iterations:
            exact = row[8] if not np.isnan(row[8]) else "N/A"
            print(f"{int(row[0]):2} | {row[1]:7.5f} | {row[2]:7.5f} | "
                  f"{row[3]:8.5f} | {row[4]:8.5f} | {row[5]:8.5f} | "
                  f"{row[6]:8.5f} | {row[7]:7.5f} | {exact:14} |")

    # ==================== МЕТОД АДАМСА ====================
    def methodAdams(self, function_number):
        try:
            runge_results = self._calculateRungeCutta4(function_number)[:4]
            adams_results = self._calculateAdams(function_number, runge_results)
            self._printAdamsTable(adams_results)
            return adams_results
        except Exception as e:
            print(f"Ошибка в методе Адамса: {str(e)}")
            return []

    def _calculateAdams(self, function_number, runge_results):
        x_arr = self.getArrayX()
        n = len(x_arr)
        h = self.getH()
        x0_val = self.getX0()
        y0_val = self.getY0()
        iterations = [[0.0] * 8 for _ in range(n)]

        # Инициализация первых 4 точек из Рунге-Кутта
        for i in range(4):
            iterations[i][0] = i
            iterations[i][1] = runge_results[i][1]
            iterations[i][2] = runge_results[i][2]
            iterations[i][3] = runge_results[i][3] / h if not np.isnan(runge_results[i][3])
else float('nan')
            if i > 0:
                iterations[i][4] = iterations[i][3] - iterations[i - 1][3] if not
np.isnan(iterations[i][3]) else float(
                    'nan')
            if i > 1:
                iterations[i][5] = iterations[i][3] - 2 * iterations[i - 1][3] + iterations[i - 2][3]
if not np.isnan(
                    iterations[i][3]) else float('nan')
            if i > 2:
                iterations[i][6] = iterations[i][3] - 3 * iterations[i - 1][3] + 3 * iterations[i -
2][3] - \
                                   iterations[i - 3][3] if not np.isnan(iterations[i][3]) else float('nan')
            iterations[i][7] = runge_results[i][8]

        # Прогноз для последующих точек
        for i in range(3, n - 1):
```

```
            try:
                # Прогноз
                y_pred = iterations[i][2] + h * (
                        iterations[i][3] +
                        (h / 2) * iterations[i][4] +
                        (5 * h ** 2 / 12) * iterations[i][5] +
                        (3 * h ** 3 / 8) * iterations[i][6]
                )

                # Коррекция
                x_next = x_arr[i + 1]
                f_next = calculateFunction(x_next, y_pred, function_number)

                # Обновление конечных разностей
                f_i = f_next
                df_i = f_i - iterations[i][3]
                d2f_i = f_i - 2 * iterations[i][3] + iterations[i - 1][3]
                d3f_i = f_i - 3 * iterations[i][3] + 3 * iterations[i - 1][3] - iterations[i - 2][3]

                # Уточнение значения y
                y_corr = iterations[i][2] + h * (
                        f_i +
                        (h / 2) * df_i +
                        (5 * h ** 2 / 12) * d2f_i +
                        (3 * h ** 3 / 8) * d3f_i
                )
            except Exception as e:
                print(f"Ошибка расчета точки {i + 1}: {str(e)}")
                y_corr = float('nan')
                f_i = df_i = d2f_i = d3f_i = float('nan')

            # Сохранение результатов
            iterations[i + 1][0] = i + 1
            iterations[i + 1][1] = round(x_next, 5)
            iterations[i + 1][2] = round(y_corr, 5)
            iterations[i + 1][3] = f_i
            iterations[i + 1][4] = df_i
            iterations[i + 1][5] = d2f_i
            iterations[i + 1][6] = d3f_i
            iterations[i + 1][7] = round(calculateAnswer(x0_val, y0_val, x_next,
function_number), 5)

        return iterations

    def _printAdamsTable(self, iterations):
        print('\n\t\tМногошаговый метод Адамса')
        print('i  |    x    |    y    |    fi    |   Δfi   |   Δ²fi   |   Δ³fi   | Точное решение |')
        for row in iterations:
            exact = row[7] if not np.isnan(row[7]) else "N/A"
            print(f"{int(row[0]):2} | {row[1]:7.5f} | {row[2]:7.5f} | {row[3]:8.5f} | "
```

```python
            f"{row[4]:7.5f} | {row[5]:7.5f} | {row[6]:7.5f} | {exact:14} |")


class Terminal:
    def work(self):
        try:
            print('\t\tЧисленное решение ОДУ')
            function_number = self.enterFunctionNumber()
            method_number = self.enterMethodNumber()
            x0 = self.enterArgument('x0')
            y0 = self.enterArgument('y0')
            xn = self.enterInterval(x0)
            h = self.enterH()
            eps = self.enterEpsilon()
            diff = DifferentialEquationsMethods(x0, y0, xn, h, eps)

            if method_number == 1:
                full_results = diff.methodEiler(function_number)
            elif method_number == 2:
                full_results = diff.methodRungeCutta4(function_number)
            else:
                full_results = diff.methodAdams(function_number)

            # Построение графика
            if full_results:
                x_values = diff.getArrayX()
                y_values = [row[2] for row in full_results]

                # Рассчет аналитического решения
                exact_solution = []
                for x in x_values:
                    try:
                        exact = calculateAnswer(x0, y0, x, function_number)
                        exact_solution.append(exact if not np.isnan(exact) else None)
                    except:
                        exact_solution.append(None)

                plt.figure(figsize=(10, 6))
                plt.plot(x_values, y_values, 'b-', label='Численное решение')

                # Фильтрация валидных точек аналитического решения
                valid_exact = [(x, y) for x, y in zip(x_values, exact_solution) if y is not
None]
                if valid_exact:
                    x_valid, y_valid = zip(*valid_exact)
                    plt.plot(x_valid, y_valid, 'r--', label='Точное решение')

                plt.xlabel('x')
                plt.ylabel('y')
                plt.title(f'Решение ОДУ ({METHODS[method_number - 1]})')
```

```
                plt.legend()
                plt.grid(True)
                plt.show()

        except IncorrectValueException as e:
            print(f"Ошибка: {e.message}")
        except Exception as e:
            print(f"Непредвиденная ошибка: {str(e)}")

    def enterInterval(self, x0):
        try:
            print('Введите границу интервала xn (x0 задано):')
            xn = Validator.validateNumber(input())
            Validator.validateBorders(x0, xn)
            return xn
        except IncorrectValueException as e:
            print(e.message)
            return self.enterInterval(x0)

    def enterH(self):
        try:
            print('Введите шаг h:')
            h = Validator.validateH()
            return h
        except IncorrectValueException as e:
            print(e.message)
            return self.enterH()

    def enterArgument(self, arg):
        try:
            print(f'Введите значение {arg}:')
            return Validator.validateNumber(input())
        except IncorrectValueException as e:
            print(e.message)
            return self.enterArgument(arg)

    def enterFunctionNumber(self):
        try:
            print('Выберите функцию:')
            for i in range(1, len(EQUATIONS) + 1):
                print(f'{i}. y\' = {EQUATIONS[i]}')
            return Validator.validateFunctionNumber()
        except IncorrectValueException as e:
            print(e.message)
            return self.enterFunctionNumber()

    def enterMethodNumber(self):
        try:
            print('Выберите метод:')
            for i, method in enumerate(METHODS, 1):
```

```
                print(f'{i}. {method}')
            return Validator.validateMethodNumber()
        except IncorrectValueException as e:
            print(e.message)
            return self.enterMethodNumber()

    def enterEpsilon(self):
        try:
            print('Введите точность epsilon:')
            return Validator.validateEpsilon(input())
        except IncorrectValueException as e:
            print(e.message)
            return self.enterEpsilon()


if __name__ == "__main__":
    warnings.filterwarnings('ignore')
    terminal = Terminal()
    while True:
        terminal.work()
        print("Хотите продолжить? (y/n)")
        if input().lower() != 'y':
            break
```