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»

Отчет по лабораторной работе №1 по курсу "Моделирование"

Тема Программная реализация различных методов при решении задачи Коши для ОДУ.

Студент Пересторонин П.Г.

Группа ИУ7-63Б

Оценка _____

Преподаватель Градов В.М.

Москва — 2021 г.

Оглавление

1 Теоретические сведения

1.1 Цель работы

Получение навыков решения задачи Коши при помощи метода Пикара, метода Эйлера и метода Рунге-Кутты 2-го порядка.

1.2 Задача

Имеем ОДУ, у которого отсутствует аналитическое решение:

$$\begin{cases} u'(x) = f(x,u) \\ u(\xi) = \eta \end{cases} \tag{1.1}$$

Для решения данного ОДУ были использованы 3 алгоритма.

1.2.1 Метод Пикара

Имеем:

$$u(x) = \eta + \int_{\xi}^{x} f(t,u(t))\,dt \tag{1.2}$$

Строим ряд функций:

$$y^{(s)} = \eta + \int_{\xi}^{x} f(t,y^{(s-1)}(t))\,dt, \qquad y^{(0)} = \eta \tag{1.3}$$

Построим 4 приближения для уравнения (1.2):

$$y^{(1)}(x) = 0 + \int_{0}^{x} t^2\,dt = \frac{x^3}{3} \tag{1.4}$$

$$y^{(2)}(x) = 0 + \int_{0}^{x} (t^2 + \left(\frac{t^3}{3}\right)^2)\,dt = \frac{x^3}{3} + \frac{x^7}{63} \tag{1.5}$$

$$y^{(3)}(x) = 0 + \int_0^x (t^2 + \left(\frac{t^3}{3} + \frac{t^7}{63}\right)^2)\, dt = \frac{x^3}{3} + \frac{x^7}{63} + \frac{2x^{11}}{2079} + \frac{x^{15}}{59535} \quad (1.6)$$

$$y^{(4)}(x) = 0 + \int_0^x (t^2 + \left(\frac{t^3}{3} + \frac{t^7}{63} + \frac{2t^{11}}{2079} + \frac{t^{15}}{59535}\right)^2)\, dt = \frac{x^3}{3} + \frac{x^7}{63} + \frac{2x^{11}}{2079} + \frac{x^{15}}{59535} + \frac{2x^{15}}{93555} + \frac{2x^{19}}{3393495} + \frac{2x^{19}}{2488563} + \frac{2x^{23}}{86266215} + \frac{x^{23}}{99411543} + \frac{2x^{27}}{3341878155} + \frac{x^{31}}{109876902975} \quad (1.7)$$

1.2.2 Метод Эйлера

$$y^{(n+1)}(x) = y^{(n)}(x) + h \cdot f(x_n, y^{(n)}) \quad (1.8)$$

Порядок точности: $O(h)$.

1.2.3 Метод Рунге-Кутта

$$y^{n+1}(x) = y^n(x) + h((1-\alpha)R_1 + \alpha R_2) \quad (1.9)$$

где $R1 = f(x_n, y^n)$, $R2 = f(x_n + \frac{h}{2\alpha}, y^n + \frac{h}{2\alpha}R_1)$, $\alpha = \frac{1}{2}$ или 1

Порядок точности: $O(h^2)$.

2 Реализация

Ниже представлены исходные коды программы на языке Elixir.

Листинг 2.1: Основной модуль приложения

```elixir
defmodule Picard.Application do
  @precision_order 4

  use Application
  import Picard.Solvers.Main
  import Picard.Solvers.Other

  def start(_, _) do
    run()
    {:ok, self()}
  end

  def run() do
    case Picard.Parser.read_input() do
      [xmax, step] ->
        Picard.Helper.float_range_generator(0, xmax + step / 2, step)
        |> (fn xlist -> {xlist, generate_picard_vals_with_names(@precision_order, xlist)}
            end).()
        |> (fn {xs, pvals} ->
              [["X" | xs] | pvals ++ generate_other_vals_with_names(xmax, step)]
            end).()
        |> Picard.Helper.pretty_print()

      _ ->
        IO.puts("Wrong input data")
    end
  end
end
```

Листинг 2.2: Функции для работы с многочленами

```elixir
defmodule Picard.Polynomial do
  @const_value 0

  def integral(polynom, const_value \\ @const_value) do
    for {degree, coeff} <- polynom,
        do: {degree + 1, coeff / (degree + 1)},
        into:
          %{}
          |> Map.put(0, const_value)
          |> clean
  end

```

```
  def clean(polynom), do: :maps.filter(fn d, c -> d >= 0 and c != 0 end, polynom)

  def sum(pol1, pol2), do: Map.merge(pol1, pol2, fn _k, v1, v2 -> v1 + v2 end)

  def pow(polynom, deg) do
    _pow(polynom, polynom, deg)
  end

  defp _pow(result, polynomial, deg) do
    cond do
      deg <= 1 ->
        result

      true ->
        _pow(mult(result, polynomial), polynomial, deg - 1)
    end
  end

  def mult(pol_1, pol_2) when is_map(pol_1) do
    pol_1
    |> Enum.flat_map(fn {d1, c1} -> Enum.map(pol_2, fn {d2, c2} -> {d1 + d2, c1 * c2}
        end) end)
    |> Enum.reduce(%{}, fn {d, c}, acc -> Map.update(acc, d, c, fn ec -> ec + c end) end)
  end

  def mult(number, pol) when is_number(number) do
    Enum.reduce(pol, %{}, fn {deg, coeff}, acc -> Map.put(acc, deg, coeff * number) end)
  end

  def value(polynom, arg) do
    Enum.reduce(polynom, 0.0, fn {deg, coeff}, acc -> coeff * :math.pow(arg, deg) + acc
        end)
  end

  def values_list(polynom, list) do
    Enum.map(list, &value(polynom, &1))
  end
end
```

Листинг 2.3: Модуль реализации метода Пикара

```
defmodule Picard.Solvers.Main do
  alias Picard.Polynomial
  @x_degree 2
  @x_coeff 1
  @y_degree 2
  @y_coeff 1

  def f(x, y), do: :math.pow(@x_coeff * x, @x_degree) + :math.pow(y, @y_degree)
```

```

  def generate_picard_vals_with_names(precision_order, xlist) do
    generate_picard_values(precision_order, xlist)
    |> Enum.with_index(1)
    |> Enum.map(fn {vals, index} -> ["Picard #{index} precision order" | vals] end)
  end

  def generate_picard_values(precision_order, xlist) do
    picard_solvers_list(precision_order)
    |> Enum.map(&Polynomial.values_list(&1, xlist))
  end

  def picard_solvers_list(precision_order) do
    Enum.reverse(picard_solvers(%{}, precision_order, []))
  end

  defp picard_solvers(polynom_y, precision_order, lst) do
    case precision_order do
      0 ->
        lst

      _ ->
        polynom_y = new_picard_solver(polynom_y)
        picard_solvers(polynom_y, precision_order - 1, [polynom_y | lst])
    end
  end

  defp new_picard_solver(polynom_y) do
    %{@x_degree => @x_coeff}
    |> Polynomial.sum(Polynomial.mult(@y_coeff, Polynomial.pow(polynom_y, @y_degree)))
    |> Polynomial.integral()
  end
end
```

Листинг 2.4: Модуль реализации методов Эйлера и Рунге-Кутта

```
defmodule Picard.Solvers.Other do
  @alpha 1
  import Picard.Helper, only: [float_range_map: 5]
  import Picard.Solvers.Main, only: [f: 2]

  def generate_other_vals_with_names(from \\ 0, to, step) do
    [
      ["Runge-Kutta method" | generate_runge_kutta_values(from, to, step, @alpha)],
      ["Euler method" | generate_euler_values(from, to, step)]
    ]
  end

  def generate_runge_kutta_values(from, to, step, alpha \\ @alpha) do
```

```
    rk_func = fn xn, ys ->
      yn = hd(ys)
      k1 = f(xn, yn)
      k2 = f(xn + step / 2 / alpha, yn + step / 2 / alpha * k1)
      yn + step * ((1 - alpha) * k1 + alpha * k2)
    end

    float_range_map(from, to + step / 2, step, [0], rk_func)
    |> Enum.reverse()
  end

  def generate_euler_values(from, to, step) do
    float_range_map(from, to + step / 2, step, [0], fn xn, ys -> hd(ys) + step * f(xn,
        hd(ys)) end)
    |> Enum.reverse()
  end
end
```

Листинг 2.5: Вспомогательный модуль чтения ввода

```
defmodule Picard.Parser do
  def read_input do
    IO.gets("  X    : ")
    |> String.trim()
    |> String.split()
    |> Enum.map(&(Float.parse(&1) |> elem(0)))
  end
end
```

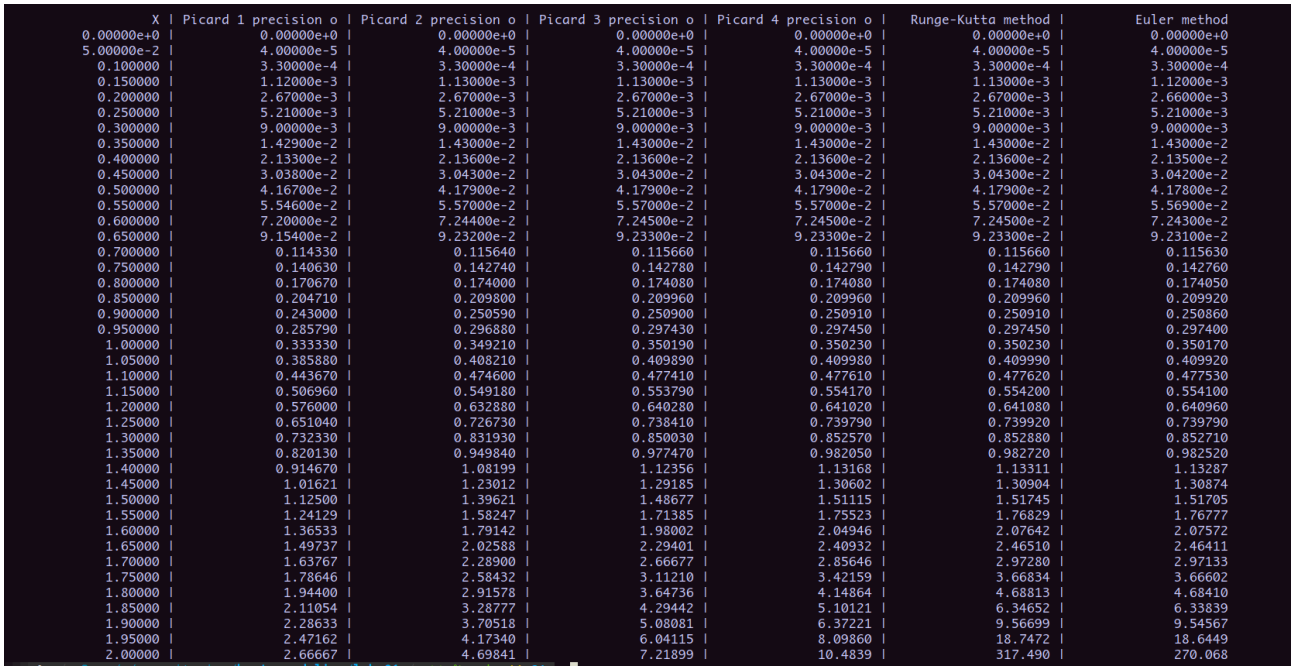
Листинг 2.6: Модуль со вспомогательными функциями

```
defmodule Picard.Helper do
  @round_precision 5
  def float_range_generator(from, to, step \\ 1) do
    float_range_map(from, to, step, [], fn n, _lst -> Float.round(n / 1,
        @round_precision) end)
    |> Enum.reverse()
  end

  def float_range_map(from, to, step, lst, func) do
    if from > to do
      lst
    else
      float_range_map(from + step, to, step, [func.(from, lst) | lst], func)
    end
  end

  def pretty_print(lst) do
    lst
```

```
    |> Enum.zip()
    |> Enum.map(&Tuple.to_list/1)
    |> Enum.map(
      &Enum.map(&1, fn elem ->
        cond do
          is_binary(elem) -> :io_lib.format("~20s", [elem])
          true -> :io_lib.format("~20g", [Float.round(elem / 1, @round_precision)])
        end
      end)
    )
    |> Enum.map(&Enum.join(&1, " | "))
    |> Enum.join("\n")
    |> IO.puts()
  end
end
```

3 Результат работы программы

Ниже приведена таблица с вычисленными значениями. Значения вычислены для шага 10^{-4}, выведено каждое 500 значение.



X	Picard 1 precision o	Picard 2 precision o	Picard 3 precision o	Picard 4 precision o	Runge-Kutta method	Euler method
0.00000e+0	0.00000e+0	0.00000e+0	0.00000e+0	0.00000e+0	0.00000e+0	0.00000e+0
5.00000e-2	4.00000e-5	4.00000e-5	4.00000e-5	4.00000e-5	4.00000e-5	4.00000e-5
0.100000	3.30000e-4	3.30000e-4	3.30000e-4	3.30000e-4	3.30000e-4	3.30000e-4
0.150000	1.12000e-3	1.13000e-3	1.13000e-3	1.13000e-3	1.13000e-3	1.12000e-3
0.200000	2.67000e-3	2.67000e-3	2.67000e-3	2.67000e-3	2.67000e-3	2.66000e-3
0.250000	5.21000e-3	5.21000e-3	5.21000e-3	5.21000e-3	5.21000e-3	5.21000e-3
0.300000	9.00000e-3	9.00000e-3	9.00000e-3	9.00000e-3	9.00000e-3	9.00000e-3
0.350000	1.42900e-2	1.43000e-2	1.43000e-2	1.43000e-2	1.43000e-2	1.43000e-2
0.400000	2.13300e-2	2.13600e-2	2.13600e-2	2.13600e-2	2.13600e-2	2.13500e-2
0.450000	3.03800e-2	3.04300e-2	3.04300e-2	3.04300e-2	3.04300e-2	3.04200e-2
0.500000	4.16700e-2	4.17900e-2	4.17900e-2	4.17900e-2	4.17900e-2	4.17800e-2
0.550000	5.54600e-2	5.57000e-2	5.57000e-2	5.57000e-2	5.57000e-2	5.56900e-2
0.600000	7.20000e-2	7.24400e-2	7.24500e-2	7.24500e-2	7.24500e-2	7.24300e-2
0.650000	9.15400e-2	9.23200e-2	9.23300e-2	9.23300e-2	9.23300e-2	9.23100e-2
0.700000	0.114330	0.115640	0.115660	0.115660	0.115660	0.115630
0.750000	0.140630	0.142740	0.142780	0.142790	0.142790	0.142760
0.800000	0.170670	0.174000	0.174080	0.174080	0.174080	0.174050
0.850000	0.204710	0.209800	0.209960	0.209960	0.209960	0.209920
0.900000	0.243000	0.250590	0.250900	0.250910	0.250910	0.250860
0.950000	0.285790	0.296880	0.297430	0.297450	0.297450	0.297400
1.00000	0.333330	0.349210	0.350190	0.350230	0.350230	0.350170
1.05000	0.385880	0.408210	0.409890	0.409980	0.409990	0.409920
1.10000	0.443670	0.474600	0.477410	0.477610	0.477620	0.477530
1.15000	0.506960	0.549180	0.553790	0.554170	0.554200	0.554100
1.20000	0.576000	0.632880	0.640280	0.641020	0.641080	0.640960
1.25000	0.651040	0.726730	0.738410	0.739790	0.739920	0.739790
1.30000	0.732330	0.831930	0.850030	0.852570	0.852880	0.852710
1.35000	0.820130	0.949840	0.977470	0.982050	0.982720	0.982520
1.40000	0.914670	1.08199	1.12356	1.13168	1.13311	1.13287
1.45000	1.01621	1.23012	1.29185	1.30602	1.30904	1.30874
1.50000	1.12500	1.39621	1.48677	1.51115	1.51745	1.51705
1.55000	1.24129	1.58247	1.71385	1.75523	1.76829	1.76777
1.60000	1.36533	1.79142	1.98002	2.04946	2.07642	2.07572
1.65000	1.49737	2.02588	2.29401	2.40932	2.46510	2.46411
1.70000	1.63767	2.28900	2.66677	2.85646	2.97280	2.97133
1.75000	1.78646	2.58432	3.11210	3.42159	3.66834	3.66602
1.80000	1.94400	2.91578	3.64736	4.14864	4.68813	4.68410
1.85000	2.11054	3.28777	4.29442	5.10121	6.34652	6.33839
1.90000	2.28633	3.70518	5.08081	6.37221	9.56699	9.54567
1.95000	2.47162	4.17340	6.04115	8.09860	18.7472	18.6449
2.00000	2.66667	4.69841	7.21899	10.4839	317.490	270.068

Рис. 3.1: Таблица с вычисленными значениями

4 Ответы на контрольные вопросы

4.1 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.

Ответ на данный вопрос даётся с опорой на значения из таблицы выше (Шаг равен 10^{-4}). Учитывая тот факт, что i-ое приближение имеет порядок точности $= O(h^i)$, можно сделать вывод, что метод Пикара i-ого порядка точности будет точен до тех пор, пока его значение совпадает (с определенной степенью точности, в нашем случае - 2 цифры после запятой) с методом Пикара $(i+1)$-ого порядка точности. Как только это значение начинает отклоняться, учитывая тот факт, что более высокое приближение имеет меньшую погрешность, метод нельзя считать решением.

Таким образом:

1. 1 приближение считается решением на полуинтервале $[0; 0.85)$, потому что в точке 0.85 его значение отличиается во 2 цифре после запятой;

2. 2 приближение считается решением на полуинтервале $[0; 1.20)$;

3. 3 приближение считается решением на полуинтервале $[0; 1.40)$;

4. 4 приближение считается решением на отрезке $[0; 1.40]$, при этом дать более точную оценку для этого интервала без наличия 5 приближения невозможно;

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

Чтобы доказать правильность полученного численными методами результата, требуется устремить шаг к 0 до тех пор, пока не получится ситуации, когда при изменении шага значение функции не будет меняться (или будет меняться незначительно, в пределах допустимых в задачи пределов). При достижении такой ситуации, полученное значение можно считать правильным.

4.3 Каково значение функции при $x = 2$, т.е. привести значение $u(2)$.

Ответ: ≈ 317.