БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Лабораторная работа №4
«Численные методы решения задачи Коши»

Выполнил
Студент 11 группы
Сергиенко Лев Эдуардович

Минск, 2024

Постановка задачи

Решить задачу Коши для обыкновенного дифференциального уравнения первого порядка на отрезке [a,b] с шагом h = 0.1 методами, указанными в варианте задания. Оценить погрешность численного решения с шагом h с помощью правила Рунге (для одношаговых методов). Сравнить полученные численные решения с точным решением u(x). В одной системе координат построить график функции u(x) и график одного из полученных численных решений.

Вариант

Методы: Неявный метод трапеций; явный метод Рунге-Кутты 3-го порядка; предиктор-корректорный метод Адамса 3-го порядка.

$$u' = \frac{1-u^2}{2x}, \quad x \in [2,3]$$

$$u(2) = 2$$

$$u(x) = \frac{3x+2}{3x-2}$$

Применяемые численные методы. Итерационный процесс метода Ньютона для реализации неявного метода трапеций

$u'(x) = f(x, u(x)), a \leq x \leq b$

$u(a) = u_0$

$x_i = a + ih$

Неявный метод трапеций

$$y_{i+1} = y_i + \frac{h}{2}(f(x_i, y_i) + f(x_{i+1}, y_{i+1}))$$

Итерационный процесс метода Ньютона для метода трапеций

$$y_{i+1}^{k+1} = y_{i+1}^{k} - \frac{y_{i+1}^{k} - y_i - \frac{h}{2}(f(x_i, y_i) + f(x_{i+1}, y_{i+1}^{k}))}{1 - \frac{h}{2}f'(x_{i+1}, y_{i+1}^{k})}$$

Явный метод Рунге-Кутты 3-го порядка

$y_{i+1} = y_i + h(\frac{1}{4}k_1 + \frac{3}{4}k_3)$

$k_1 = f(x_i, y_i)$

$k_2 = f(x_i + \frac{1}{3}h, y_i + \frac{1}{3}k_1 h)$

$k_3 = f(x_i + \frac{2}{3}h, y_i + \frac{2}{3}k_2 h)$

Предиктор-корректорный метод Адамса 3-го порядка

Явный метод Адамса 3-го порядка:

$y_{i+1} = y_i + \frac{h}{12}(23f(x_i, y_i) - 16f(x_{i-1}, y_{i-1}), 5f(x_{i-2}, y_{i-2}))$, где начало таблицы y_{i-1}, y_i вычисляется с помощью явного метода Рунге-Кутты 3-го порядка.

Неявный метод Адамса 3-го порядка:

$$y_{i+1} = y_i + \frac{h}{12}(5f(x_{i+1}, y_{i+1}^{explicit}) + 8f(x_i, y_i) - f(x_{i-1}, y_{i-1}))$$

Правило Рунге оценки погрешности

$2h = 0.2$
$h = 0.1$

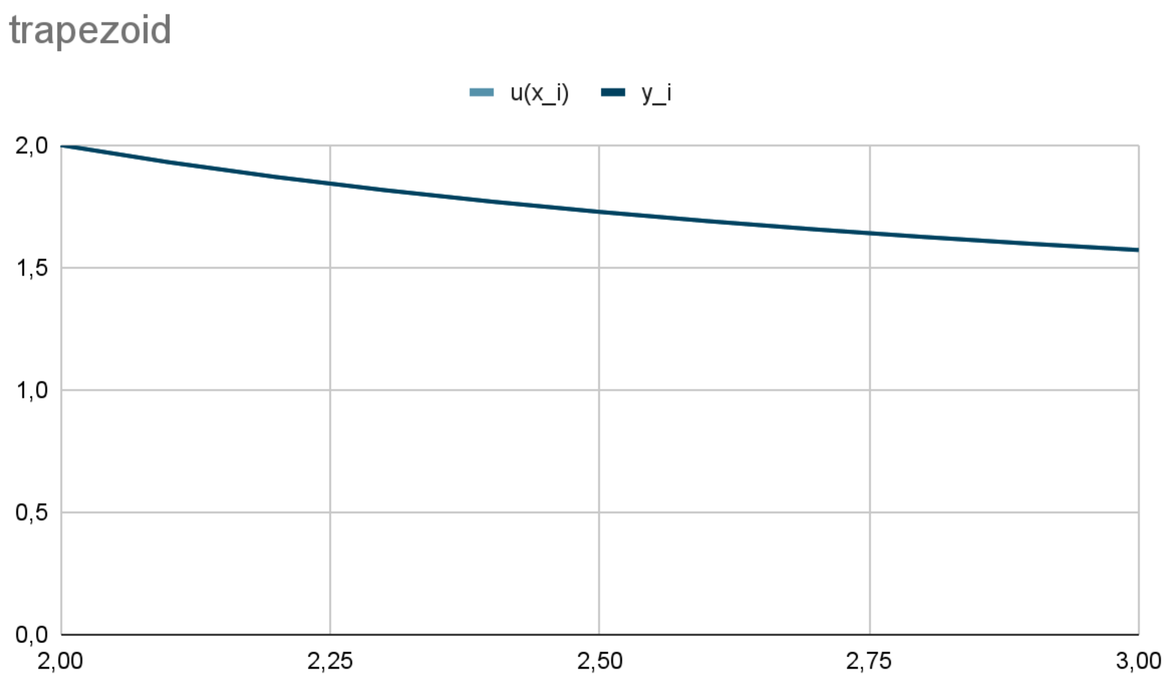
$R_h = \dfrac{\max_i |y_h(x_{2i}) - y_{2h}(x_i)|}{2^p - 1}$, где p - порядок точности одношагового метода

p = 3 - для Рунге-Кутты 3-го порядка
p = 2 - для неявного метода трапеций

Результаты вычислительного эксперимента, оформленные в виде таблицы

i	x_i	точное решение	Численное решение задачи Коши с шагом 0.1		
			Метод 1	Метод 2	Метод 3
		u(x_i)	y_i	y_i	y_i
0	2	2.0	2.0000000000	2.0000000000	2.0000000000
1	2.1	1.9302325581	1.9300580469	1.9302247867	1.9302247867
2	2.2	1.8695652174	1.8692731166	1.8695526991	1.8695526991
3	2.3	1.8163265306	1.8159552119	1.8163111469	1.8163359293
4	2.4	1.7692307692	1.7688065690	1.7692137141	1.7692542961
5	2.5	1.7272727273	1.7268139881	1.7272547697	1.7273054261
6	2.6	1.6896551724	1.6891748190	1.6896368128	1.6896937844
7	2.7	1.6557377049	1.6552449251	1.6557192703	1.6557800326
8	2.8	1.6250	1.6245013643	1.6249817037	1.6250445438
9	2.9	1.5970149254	1.5965151472	1.5969969052	1.5970606533
10	3	1.5714285714	1.5709310313	1.5714109145	1.5714747693
max |u(x_i) - y_i|			0.0004997781	0.0000184346	0.0000461979
Оценка погрешности по правилу Рунге			0.0005009795	0.0000199274	-

График функции u(x) и график одного из полученных численных решений



Выводы

1. Точность методов:

- На основании полученных результатов можно сделать вывод, что все три численных метода (неявный метод трапеций, явный метод Рунге-Кутты 3-го порядка и предиктор-корректорный метод Адамса 3-го порядка) показали высокую точность решения задачи Коши.
- Наибольшую точность показал явный метод Рунге-Кутты 3-го порядка, с максимальным отклонением от точного решения 0.0000184346.
- Неявный метод трапеций продемонстрировал максимальное отклонение 0.0004997781.
- Предиктор-корректорный метод Адамса 3-го порядка занял промежуточное место по точности, с максимальным отклонением 0.0000461979.

2. Оценка погрешности по правилу Рунге:

- Правило Рунге оценило погрешность для неявного метода трапеций на уровне 0.0005009795, что близко к фактической максимальной ошибке.
- Для явного метода Рунге-Кутты 3-го порядка оценка погрешности по правилу Рунге составила 0.0000199274, что также подтверждает его высокую точность.
- Для предиктор-корректорного метода Адамса 3-го порядка оценка погрешности не была предоставлена в таблице, но по фактическим данным метод также демонстрирует высокую точность.

3. Графическое представление:

- Построенные графики функции u(x) и численных решений показывают, что численные методы очень точно аппроксимируют точное решение на заданном отрезке [2, 3] с шагом 0.1.

- Визуально все три метода дают решения, практически совпадающие с точным решением, что подтверждает их высокую эффективность и точность для данной задачи.

4. Практическая применимость:

- Явный метод Рунге-Кутты 3-го порядка является предпочтительным для использования в задачах, требующих высокой точности и быстрого вычисления.
- Неявный метод трапеций, хотя и менее точен, может быть полезен в задачах, где устойчивость метода играет ключевую роль.

5. Заключение:

- Все исследованные методы являются подходящими для решения задачи Коши для обыкновенного дифференциального уравнения первого порядка.
- Явный метод Рунге-Кутты 3-го порядка оказался наиболее точным и рекомендован к применению в случаях, когда важна высокая точность численного решения.
- Выбор метода должен зависеть от конкретных требований задачи, таких как требуемая точность и устойчивость решения.

Листинг программы с комментариями

main.go:

```
package main

import (
    "mv2.4/diffs"
)

const (
    a  = 2.0
    b  = 3.0
    h  = 0.1
    u0 = 2.0
)

var (
    diffU = func(x float64, u float64) float64 {
        return (1.0 - u*u) / (2.0 * x)
    }

    u = func(x float64) float64 {
        return (3.0*x + 2.0) / (3.0*x - 2.0)
    }

    ddU = func(x float64, u float64) float64 {
        return (1.0 - 2.0*u) / (2.0 * x)
    }
)

// const (
//  a  = 0.0 // начальное значение x
//  b  = 1.0 // конечное значение x
//  h  = 0.1 // шаг интегрирования
//  y0 = 1.0 // начальное значение y
// )

// var (
//  diffY = func(x float64, y float64) float64 {
//      return -15.0 * y
//  }

//  yExact = func(x float64) float64 {
```

```
//      return math.Exp(-15.0 * x)
//  }

//  ddY = func(x float64, y float64) float64 {
//      return -15.0
//  }
// )

func main() {
    task := diffs.NewCoshieTask(diffU, u, a, b, u0, h)

    task.SolveRungeKutta3("runge.txt")
    task.ImplicitTrapezoid("trapezoid.txt", ddU)
    task.AdamsPredictorCorrector3("adams.txt")
}
```

cohie.go:

```
package diffs

import (
    "fmt"
    "log"
    "math"
    "os"
)

const (
    e = 1e-7 // Допустимая погрешность для метода Ньютона
)

// Правая часть дифференциального уравнения
type diffEquation func(x float64, u float64) float64

// Функция u(x)
type equation func(x float64) float64

type CoshieTask struct {
    diffU diffEquation // Правая часть дифференциального уравнения
    u     equation     // Функция u(x)

    a  float64
    b  float64
    u0 float64 // Начальное условие
```

```
    h float64 // Шаг
}

func NewCoshieTask(diffU diffEquation, u equation, a, b, u0, h float64)
CoshieTask {
    return CoshieTask{
        diffU: diffU,
        u:     u,
        a:     a,
        b:     b,
        u0:    u0,
        h:     h,
    }
}

// SolveRungeKutta3 решает дифференциальное уравнение методом
Рунге-Кутта 3-го порядка
// и записывает результат в файл с именем fileName
func (t CoshieTask) SolveRungeKutta3(fileName string) {
    file, err := os.OpenFile(fileName,
os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0644)
    if err != nil {
        log.Fatal(err)
    }
    defer file.Close()

    yi := t.u0
    yih2 := t.u0
    maxErr := 0.0
    Rh := 0.0

    fmt.Fprintf(file, "i = %.10f u = %.10f y = %.10f\n", t.a, t.u(t.a),
yi)

    k := 1
    for xi := t.a; xi < t.b; xi += t.h {
        yi1 := rungeKutta3(t.diffU, xi, yi, t.h)

        fmt.Fprintf(file, "i = %.10f u = %.10f y = %.10f\n", xi+t.h,
t.u(xi+t.h), yi1)

        maxErr = max(math.Abs(t.u(xi+t.h)-yi1), maxErr)
```

```go
        // Считаем погрешность методом Рунге для шага 2h
        if k%2 == 0 {
            yih2 = rungeKutta3(t.diffU, xi-t.h, yih2, t.h*2.0)
            Rh = max(math.Abs(yih2-yi1)/7.0, Rh)
        }

        k++
        yi = yi1
    }
    fmt.Fprintf(file, "err            = %.10f\n", maxErr)
    fmt.Fprintf(file, "Rh             = %.10f\n", Rh)
    fmt.Fprintf(file, "|Rh - err|     = %.10f\n", math.Abs(Rh-maxErr))
}

// Основная логика метода Рунге-Кутта 3-го порядка
func rungeKutta3(diffU diffEquation, xi, yi, h float64) float64 {
    var (
        k1 = diffU(xi, yi)
        k2 = diffU(xi+1.0/3.0*h, yi+1.0/3.0*k1*h)
        k3 = diffU(xi+2.0/3.0*h, yi+2.0/3.0*k2*h)
    )

    return yi + h*(1.0/4.0*k1+3.0/4.0*k3)
}

// ImplicitTrapezoid решает дифференциальное уравнение неявным методом
трапеций
// ddU - производная правой части дифференциального уравнения для
метода Ньютона
func (t CoshieTask) ImplicitTrapezoid(fileName string, ddU
diffEquation) {
    file, err := os.OpenFile(fileName,
os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0644)
    if err != nil {
        log.Fatal(err)
    }
    defer file.Close()

    yi := t.u0
    yih2 := t.u0
    maxErr := 0.0
    Rh := 0.0
```

```
    fmt.Fprintf(file, "i = %.10f u = %.10f y = %.10f\n", t.a, t.u(t.a),
yi)

    k := 1
    for xi := t.a; xi < t.b; xi += t.h {
        yi1 := implicitTrapezoid(t.diffU, ddU, xi, yi, t.h)

        fmt.Fprintf(file, "i = %.10f u = %.10f y = %.10f\n", xi+t.h,
t.u(xi+t.h), yi1)

        maxErr = max(math.Abs(t.u(xi+t.h)-yi1), maxErr)
        // Считаем погрешность методом Рунге для шага 2h
        if k%2 == 0 {
            yih2 = implicitTrapezoid(t.diffU, ddU, xi-t.h, yih2,
t.h*2.0)
            Rh = max(math.Abs(yih2-yi1)/3.0, Rh)
        }

        yi = yi1
        k++
    }
    fmt.Fprintf(file, "err          = %.10f\n", maxErr)
    fmt.Fprintf(file, "Rh           = %.10f\n", Rh)
    fmt.Fprintf(file, "|Rh - err|   = %.10f\n", math.Abs(Rh-maxErr))
}

// Основная логика неявного метода трапеций
func implicitTrapezoid(diffU diffEquation, ddU diffEquation, xi, yi, h
float64) float64 {
    return yi + h/2.0*(diffU(xi, yi)+diffU(xi+h, newtonNonlinear(diffU,
ddU, xi, yi, h)))
}

// newtonNonlinear решает нелинейное уравнение методом Ньютона
func newtonNonlinear(diffU diffEquation, ddU diffEquation, xi, yi, h
float64) float64 {
    var (
        y           = yi
        hh          = h / 2.0
        xi1         = xi + h
        precomputed = yi + hh*diffU(xi, yi)
    )
```

```
    for {
        yNext := y - (y-precomputed-hh*diffU(xi1, y))/(1.0-hh*ddU(xi1,
y))
        if math.Abs(yNext-y) < e {
            return yNext
        }
        y = yNext
    }
}

// AdamsPredictorCorrector3 решает дифференциальное уравнение
предиктор-корректорным методом Адамса 3-го порядка
func (t CoshieTask) AdamsPredictorCorrector3(fileName string) {
    file, err := os.OpenFile(fileName,
os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0644)
    if err != nil {
        log.Fatal(err)
    }
    defer file.Close()

    yi_2 := t.u0
    yi_1 := rungeKutta3(t.diffU, t.a, yi_2, t.h)   // находим y[i-1]
методом Рунге-Кутта 3-го порядка
    yi := rungeKutta3(t.diffU, t.a+t.h, yi_1, t.h) // находим y[i]
методом Рунге-Кутта 3-го порядка
    maxErr := 0.0

    fmt.Fprintf(file, "i = %.10f u = %.10f y = %.10f\n", t.a, t.u(t.a),
yi_2)
    fmt.Fprintf(file, "i = %.10f u = %.10f y = %.10f\n", t.a+t.h,
t.u(t.a+t.h), yi_1)
    fmt.Fprintf(file, "i = %.10f u = %.10f y = %.10f\n", t.a+2.0*t.h,
t.u(t.a+2.0*t.h), yi)

    maxErr = max(math.Abs(t.u(t.a+t.h)-yi_1),
math.Abs(t.u(t.a+2*t.h)-yi))

    for xi := t.a + 2.0*t.h; xi <= t.b; xi += t.h {
        yi1 := explicitAdams3(t.diffU, xi, yi, yi_1, yi_2, t.h)
        yi1 = implicitAdams3(t.diffU, xi, yi1, yi, yi_1, t.h)

        fmt.Fprintf(file, "i = %.10f u = %.10f y = %.10f\n", xi+t.h,
t.u(xi+t.h), yi1)
```

```
        maxErr = max(math.Abs(t.u(xi+t.h)-yi1), maxErr)

        yi_2 = yi_1
        yi_1 = yi
        yi = yi1
    }
    fmt.Fprintf(file, "err          = %.10f", maxErr)
}

// Явный метод Адамса 3-го порядка
// yi_1 - y[i-1], yi_2 - y[i-2]
func explicitAdams3(diffU diffEquation, xi, yi, yi_1, yi_2, h float64)
float64 {
    return yi + h/12.0*(23.0*diffU(xi, yi)-16.0*diffU(xi-h,
yi_1)+5.0*diffU(xi-2*h, yi_2))
}

// Неявный метод Адамса 3-го порядка
// yi1 - y[i+1], yi_1 - y[i-1]
func implicitAdams3(diffU diffEquation, xi, yi1, yi, yi_1, h float64)
float64 {
    return yi + h/12.0*(5.0*diffU(xi+h, yi1)+8.0*diffU(xi,
yi)-diffU(xi-h, yi_1))
}
```