Московский авиационный институт
(Национальный Исследовательский Институт)

Институт №8 информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №3
по курсу «Численные методы»

Студент: Суханов Е.А

Группа: М8О-406Б-19

Оценка:

Преподаватель: Ревизников Д.Л.

Подпись:

Москва
2022

Задание 3.1

Используя таблицу значений __ функции __, вычисленных в точках __ построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки ___. Вычислить значение погрешности интерполяции в точке __.

Вариант:

$y = \log(x)$

0.1,-2.30259

0.5,-0.69315

0.9,-0.10536

1.3,0.26236

Листинг

main.go:

```
package main

import (
  "encoding/csv"
  "fmt"
  "math"
  "os"

  "gonum.org/v1/plot"
  "gonum.org/v1/plot/plotter"
  "gonum.org/v1/plot/plotutil"
)

type Point struct {
  x, y float64
}

type ftype func(float64) float64

func MakeLangrangeInterpolation(points []Point) func(float64) float64 {
  n := len(points)
  l := make([]float64, n)
  for i := 0; i < n; i++ {
    var w float64 = 1
    for j := 0; j < n; j++ {
      if j == i {
        continue
      }
      w *= points[i].x - points[j].x
```

```
    }

    l[i] = points[i].y / w
  }

  return func(x float64) float64 {
    var res float64
    for i := 0; i < n; i++ {
      var xx float64 = 1
      for j := 0; j < n; j++ {
        if j == i {
          continue
        }
        xx *= x - points[j].x
      }
      res += l[i] * xx
    }
    return res
  }
}

func MakeNewtonInterpolation(points []Point) func(float64) float64 {
  n := len(points)
  mat := make([][]float64, n)
  mat[0] = make([]float64, n)
  for i := 0; i < n; i++ {
    mat[0][i] = points[i].y
  }

  for i := 1; i < n; i++ {
    mat[i] = make([]float64, n-i)
    for j := i; j < n; j++ {
      mat[i][j-i] = (mat[i-1][j-i] - mat[i-1][j-i+1]) / (points[j-i].x -
points[j].x)
    }
  }

  return func(x float64) float64 {
    ans := float64(0)
    pow := float64(1)
    for i := 0; i < n; i++ {
      ans += pow * mat[i][0]
      pow *= x - points[i].x
    }
    return ans
  }
}

func f(x float64) float64 {
  return math.Log(x)
}
```

```
func readFromFile(filePath string) []Point {
  f, err := os.Open(filePath)
  if err != nil {
    panic("Unable to read input file " + filePath + " " + err.Error())
  }
  defer f.Close()

  csvReader := csv.NewReader(f)
  records, err := csvReader.ReadAll()
  if err != nil {
    panic("Unable to parse file as CSV for " + filePath + " " + err.Error())
  }

  points := make([]Point, len(records))
  for i := 0; i < len(records); i++ {
    _, err := fmt.Sscanf(records[i][0], "%f", &points[i].x)
    if err != nil {
      panic(err.Error())
    }
    _, err = fmt.Sscanf(records[i][1], "%f", &points[i].y)
    if err != nil {
      panic(err.Error())
    }
  }
  return points
}

func genPlot(path string, lf ftype, nf ftype, f ftype, a float64, b float64, h
float64) {
  p := plot.New()

  p.Title.Text = "Interpolation"
  p.X.Label.Text = "X"
  p.Y.Label.Text = "Y"

  steps := int((b - a) / h)
  o_p := make(plotter.XYs, steps)
  l_p := make(plotter.XYs, steps)
  n_p := make(plotter.XYs, steps)
  x := a
  for step := 0; step < steps; step++ {
    o_p[step].X = x
    o_p[step].Y = f(x)
    l_p[step].X = x
    l_p[step].Y = lf(x)
    n_p[step].X = x
    n_p[step].Y = nf(x)

    x += h
  }
```

```
  err := plotutil.AddLinePoints(p,
    "Original", o_p,
    "Langrange", l_p,
    "Newton", n_p)
  if err != nil {
    panic(err)
  }

  // Save the plot to a PNG file.
  if err := p.Save(2000, 2000, path); err != nil {
    panic(err)
  }
}

func main() {
  if len(os.Args) < 2 {
    panic("Аргументов должно быть два")
  }
  inputFile := os.Args[1]
  outputFile := os.Args[2]

  points := readFromFile(inputFile)
  lf, nf := MakeLangrangeInterpolation(points), MakeNewtonInterpolation(points)
  eps := math.Abs(f(0.8) - lf(0.8))

  fmt.Println("Значение интерполяционного многочлена: ", lf(0.8), "Значение
погрешности: ", eps)
  fmt.Println("Значение интерполяционного многочлена: ", nf(0.8), "Значение
погрешности: ", eps)

  genPlot(outputFile, lf, nf, f, points[0].x, points[len(points)-1].x, 0.01)
}
```

Вывод программы

```
Значение интерполяционного многочлена:  -0.20036421874999993 Значение
погрешности:  0.022779332564209775

Значение интерполяционного многочлена:  -0.20036421875000013 Значение
погрешности:  0.022779332564209775
```

Задание 3.2

Построить кубический сплайн для функции, заданной в узлах интерполяции. Вычислить значение функции в точке 1.5.

Вариант:

X = 1.5

0,0

1,1.8415

2,2.9093

3,3.1411

4,3.2432

Листинг

main.go:

```
package main

import (
  "encoding/csv"
  "fmt"
  "os"

  "github.com/Reterer/number_methods/internal/run_through"
  "github.com/Reterer/number_methods/internal/utils"
  "github.com/Reterer/number_methods/pkg/matrix"
  "gonum.org/v1/plot"
  "gonum.org/v1/plot/plotter"
  "gonum.org/v1/plot/plotutil"
)

type Point struct {
  x, y float64
}

type ftype func(float64) float64

func MakeSplainInterpolation(points []Point) func(float64) float64 {
  n := len(points) - 1

  c := make([]float64, n)
  {
    mat := matrix.MakeRealMatrix(n-1, n-1)
    b := matrix.MakeRealMatrix(n-1, 1)
```

```
    for i := 0; i < n-1; i++ {
      hc := points[i+2].x - points[i+1].x
      hp := points[i+1].x - points[i].x

      if i > 0 {
        mat.SetEl(i, i-1, hp)
      }

      mat.SetEl(i, i, 2*(hp+hc))

      if i < n-2 {
        mat.SetEl(i, i+1, hp)
      }

      fc := points[i+2].y - points[i+1].y
      fp := points[i+1].y - points[i].y
      b.SetEl(i, 0, 3*(fc/hc-fp/hp))

    }

    utils.PrintMatrix(mat)
    utils.PrintMatrix(b)
    c_2n := run_through.Do(mat, b)
    utils.PrintMatrix(c_2n)

    for i := 0; i < n-1; i++ {
      c[i+1] = c_2n.GetEl(i, 0)
    }
  }

  a := make([]float64, n)
  for i := 0; i < n; i++ {
    a[i] = points[i].y
  }

  b := make([]float64, n)
  for i := 0; i < n-1; i++ {
    fcurr := points[i+1].y - points[i].y
    hcurr := points[i+1].x - points[i].x
    b[i] = fcurr/hcurr - 1./3.*hcurr*(c[i+1]+2*c[i])
  }
  b[n-1] = (points[n].y-points[n-1].y)/(points[n].x-points[n-1].x) -
2./3.*(points[n].x-points[n-1].x)*c[n-1]

  d := make([]float64, n)
  for i := 0; i < n-1; i++ {
    hcurr := points[i+1].x - points[i].x
    d[i] = (c[i+1] - c[i]) / (3 * hcurr)
  }
  d[n-1] = -c[n-1] / (3 * (points[n].x - points[n-1].x))
```

```
  fmt.Println("A: ", a)
  fmt.Println("B: ", b)
  fmt.Println("C: ", c)
  fmt.Println("D: ", d)

  return func(x float64) float64 {
    // find interval
    i := 0
    for ; points[i+1].x < x; i++ {
    }
    dx := x - points[i].x
    return a[i] + b[i]*dx + c[i]*dx*dx + d[i]*dx*dx*dx
  }
}

func readFromFile(filePath string) []Point {
  f, err := os.Open(filePath)
  if err != nil {
    panic("Unable to read input file " + filePath + " " + err.Error())
  }
  defer f.Close()

  csvReader := csv.NewReader(f)
  records, err := csvReader.ReadAll()
  if err != nil {
    panic("Unable to parse file as CSV for " + filePath + " " + err.Error())
  }

  points := make([]Point, len(records))
  for i := 0; i < len(records); i++ {
    _, err := fmt.Sscanf(records[i][0], "%f", &points[i].x)
    if err != nil {
      panic(err.Error())
    }
    _, err = fmt.Sscanf(records[i][1], "%f", &points[i].y)
    if err != nil {
      panic(err.Error())
    }
  }
  return points
}

func genPlot(path string, sf ftype, points []Point, a float64, b float64, h
float64) {
  p := plot.New()

  p.Title.Text = "Interpolation"
  p.X.Label.Text = "X"
  p.Y.Label.Text = "Y"

  steps := int((b - a) / h)
```

```
  s_p := make(plotter.XYs, steps)
  x := a
  for step := 0; step < steps; step++ {
    s_p[step].X = x
    s_p[step].Y = sf(x)

    x += h
  }
  err := plotutil.AddLinePoints(p,
    "Splain", s_p)
  if err != nil {
    panic(err)
  }

  // Scatter
  scatter_data := make(plotter.XYs, len(points))
  for i := 0; i < len(points); i++ {
    scatter_data[i].X = points[i].x
    scatter_data[i].Y = points[i].y
  }
  s, err := plotter.NewScatter(scatter_data)
  if err != nil {
    panic(err)
  }
  s.GlyphStyle.Radius = 10
  p.Add(s)
  p.Legend.Add("Points", s)
  // Save the plot to a PNG file.
  if err := p.Save(2000, 2000, path); err != nil {
    panic(err)
  }
}

func main() {
  if len(os.Args) < 2 {
    panic("Аргументов должно быть два")
  }
  inputFile := os.Args[1]
  outputFile := os.Args[2]

  points := readFromFile(inputFile)

  sf := MakeSplainInterpolation(points)
  // eps := math.Abs(f(0.8) - lf(0.8))
  fmt.Println("Значение интерполяционного многочлена: ", sf(1.5))
  genPlot(outputFile, sf, points, points[0].x, points[len(points)-1].x, 0.1)

}
```

Вывод программы

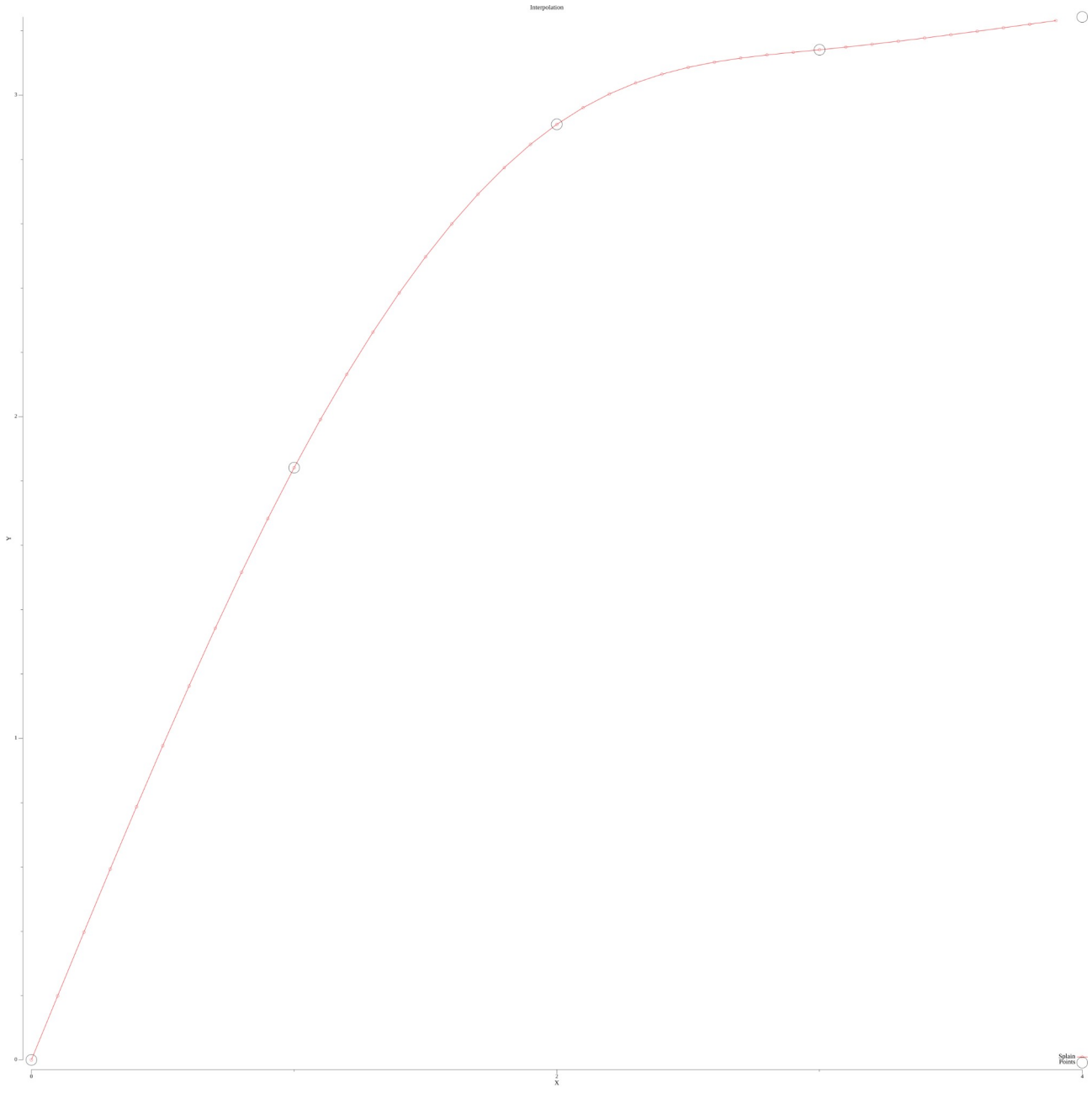
A: [0 1.8415 2.9093 3.1411]

B: [1.991342857142857 1.5418142857142856 0.5692999999999999 0.07978571428571415]

C: [0 -0.4495285714285712 -0.5229857142857146 0.03347142857142889]

D: [-0.14984285714285708 -0.02448571428571446 0.1854857142857145 -0.011157142857142963]

Значение интерполяционного многочлена: 2.4969642857142857



Задание 3.3

Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

Вариант:

0,0

1.7,1.3038

3.4,1.8439

5.1,2.2583

6.8,2.6077

8.5,2.9155

Листинг

main.go:

```
package main

import (
  "encoding/csv"
  "fmt"
  "math"
  "os"

  "github.com/Reterer/number_methods/internal/lu_decompose"
  "github.com/Reterer/number_methods/pkg/matrix"
  "gonum.org/v1/plot"
  "gonum.org/v1/plot/plotter"
  "gonum.org/v1/plot/plotutil"
)

type Point struct {
  x, y float64
}
type ftype func(float64) float64

func squareError(points []Point, f func(float64) float64) float64 {
  var err float64

  for i := 0; i < len(points); i++ {
    err += math.Pow(f(points[i].x)-points[i].y, 2)
```

```
  }

  return err
}

func lsm(points []Point, n int) func(float64) float64 {
  N := len(points)
  n1 := n + 1
  a := make([]float64, n1)

  {
    // Делаем систему
    A := matrix.MakeRealMatrix(n1, n1)
    b := matrix.MakeRealMatrix(n1, 1)

    for k := 0; k < n1; k++ {
      for i := 0; i < n1; i++ {
        var sumA, sumB float64
        for j := 0; j < N; j++ {
          sumA += math.Pow(points[j].x, float64(k+i))
          sumB += math.Pow(points[j].x, float64(k)) * points[j].y
        }
        A.SetEl(k, i, sumA)
        b.SetEl(k, 0, sumB)
      }
    }
    // Решаем систему
    LU := lu_decompose.MakeLU(lu_decompose.PermMin, A)
    LU.Decompose()
    aMat := LU.Solve(b)
    for i := 0; i < n1; i++ {
      a[i] = aMat.GetEl(i, 0)
    }
  }
  return func(x float64) float64 {
    var ans float64
    var xk float64 = 1

    for i := 0; i < n1; i++ {
      ans += xk * a[i]
      xk *= x
    }

    return ans
  }
}

func readFromFile(filePath string) []Point {
  f, err := os.Open(filePath)
  if err != nil {
    panic("Unable to read input file " + filePath + " " + err.Error())
```

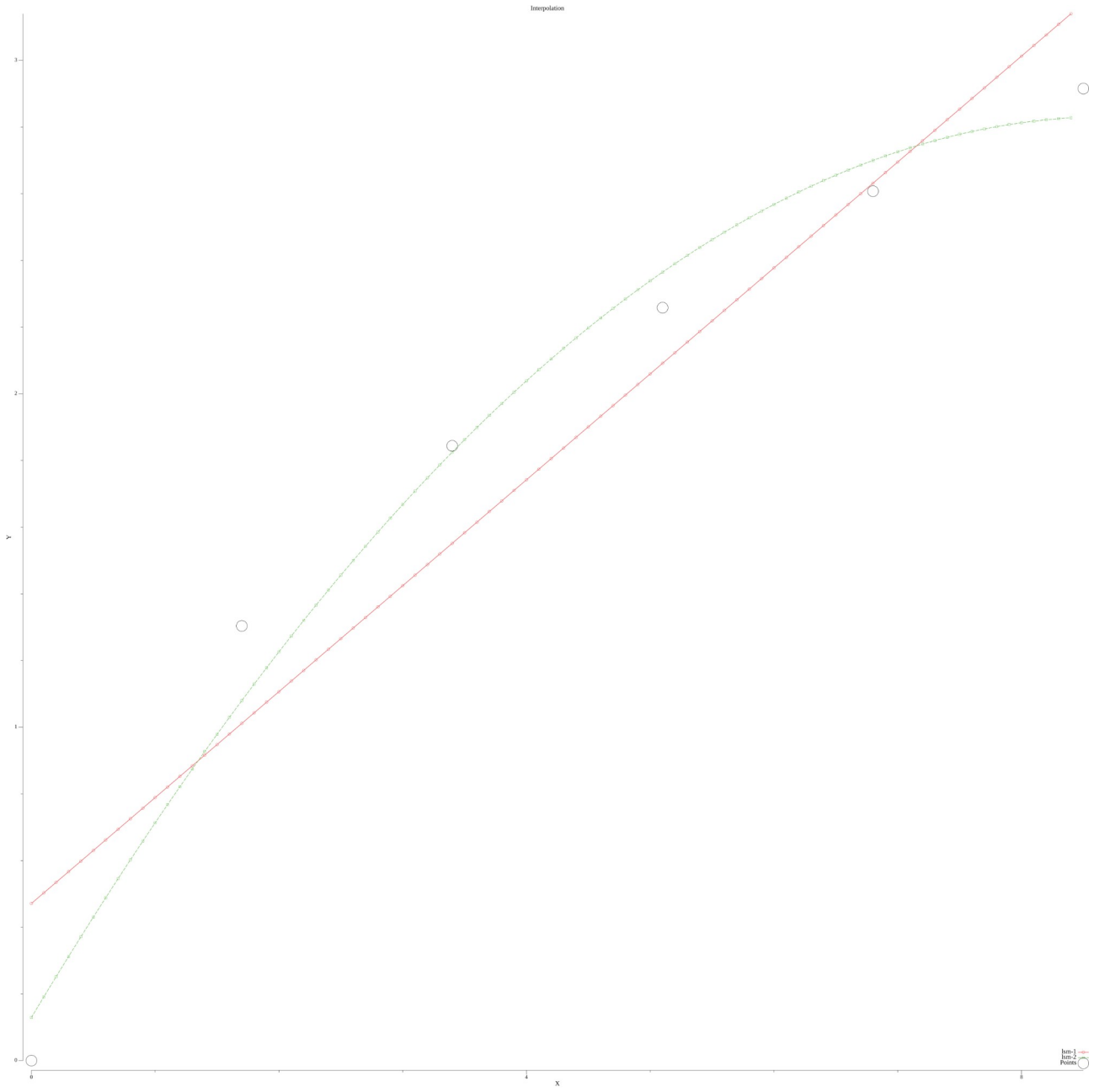
```
  }
  defer f.Close()

  csvReader := csv.NewReader(f)
  records, err := csvReader.ReadAll()
  if err != nil {
    panic("Unable to parse file as CSV for " + filePath + " " + err.Error())
  }

  points := make([]Point, len(records))
  for i := 0; i < len(records); i++ {
    _, err := fmt.Sscanf(records[i][0], "%f", &points[i].x)
    if err != nil {
      panic(err.Error())
    }
    _, err = fmt.Sscanf(records[i][1], "%f", &points[i].y)
    if err != nil {
      panic(err.Error())
    }
  }
  return points
}

func genPlot(path string, f_1 ftype, f_2 ftype, points []Point, a float64, b
float64, h float64) {
  p := plot.New()

  p.Title.Text = "Interpolation"
  p.X.Label.Text = "X"
  p.Y.Label.Text = "Y"

  steps := int((b - a) / h)
  f1_p := make(plotter.XYs, steps)
  f2_p := make(plotter.XYs, steps)
  x := a
  for step := 0; step < steps; step++ {
    f1_p[step].X = x
    f1_p[step].Y = f_1(x)
    f2_p[step].X = x
    f2_p[step].Y = f_2(x)

    x += h
  }
  err := plotutil.AddLinePoints(p,
    "lsm-1", f1_p,
    "lsm-2", f2_p,
  )
  if err != nil {
    panic(err)
  }
```

```
  // Scatter
  scatter_data := make(plotter.XYs, len(points))
  for i := 0; i < len(points); i++ {
    scatter_data[i].X = points[i].x
    scatter_data[i].Y = points[i].y
  }
  s, err := plotter.NewScatter(scatter_data)
  if err != nil {
    panic(err)
  }
  s.GlyphStyle.Radius = 10
  p.Add(s)
  p.Legend.Add("Points", s)
  // Save the plot to a PNG file.
  if err := p.Save(2000, 2000, path); err != nil {
    panic(err)
  }
}

func main() {
  if len(os.Args) < 2 {
    panic("Аргументов должно быть два")
  }
  inputFile := os.Args[1]
  outputFile := os.Args[2]
  points := readFromFile(inputFile)

  f_1 := lsm(points, 1)
  fmt.Println("a + bx: ", f_1(5.1), " serr: ", squareError(points, f_1))

  f_2 := lsm(points, 2)
  fmt.Println("a + bx + cx^2: ", f_2(5.1), " serr: ", squareError(points, f_2))

  genPlot(outputFile, f_1, f_2, points, points[0].x, points[len(points)-1].x, 0.1)

}
```

Вывод программы

a + bx: 2.0915847619047616 serr: 0.48717378819047635

a + bx + cx^2: 2.3650514285714332 serr: 0.09455769485714287


Interpolation
3
2
1
0
Y
0
4
8
X
lstm-1
lstm-2
Points

Задание 3.4

Вычислить первую и вторую производную от таблично заданной функции в точке 0.2.

Вариант:

0,1

0.1,1.1052

0.2,1.2214

0.3,1.3499

0.4,1.3499

Листинг

main.go:

```
package main

import (
  "encoding/csv"
  "fmt"
  "os"

  "gonum.org/v1/plot"
  "gonum.org/v1/plot/plotter"
  "gonum.org/v1/plot/plotutil"
)

type Point struct {
  x, y float64
}
type ftype func(float64) float64

func firstDerivative(points []Point) func(float64) float64 {
  n := len(points) - 2
  a := make([]float64, n)
  b := make([]float64, n)

  for i := 0; i < n; i++ {
    dy1 := points[i+1].y - points[i].y
    dy2 := points[i+2].y - points[i+1].y
    dx1 := points[i+1].x - points[i].x
    dx2 := points[i+2].x - points[i+1].x
    dxm := points[i+2].x - points[i].x

    a[i] = dy1 / dx1
```

```
    b[i] = (dy2/dx2 - dy1/dx1) / dxm
  }
  return func(x float64) float64 {
    i := 0
    for ; points[i+1].x < x; i++ {
    }
    return a[i] + b[i]*(2*x-points[i].x-points[i+1].x)
  }
}

func secondDerivative(points []Point) func(float64) float64 {
  n := len(points) - 2
  a := make([]float64, n)

  for i := 0; i < n; i++ {
    dy1 := points[i+1].y - points[i].y
    dy2 := points[i+2].y - points[i+1].y
    dx1 := points[i+1].x - points[i].x
    dx2 := points[i+2].x - points[i+1].x
    dxm := points[i+2].x - points[i].x

    a[i] = 2 * (dy2/dx2 - dy1/dx1) / dxm
  }

  return func(x float64) float64 {
    i := 0
    for ; points[i+1].x < x; i++ {
    }
    return a[i]
  }
}

func readFromFile(filePath string) []Point {
  f, err := os.Open(filePath)
  if err != nil {
    panic("Unable to read input file " + filePath + " " + err.Error())
  }
  defer f.Close()

  csvReader := csv.NewReader(f)
  records, err := csvReader.ReadAll()
  if err != nil {
    panic("Unable to parse file as CSV for " + filePath + " " + err.Error())
  }

  points := make([]Point, len(records))
  for i := 0; i < len(records); i++ {
    _, err := fmt.Sscanf(records[i][0], "%f", &points[i].x)
    if err != nil {
      panic(err.Error())
    }
```

```
    _, err = fmt.Sscanf(records[i][1], "%f", &points[i].y)
    if err != nil {
      panic(err.Error())
    }
  }
  return points
}

func genPlot(path string, f ftype, ff ftype, points []Point, a float64, b float64,
h float64) {
  p := plot.New()

  p.Title.Text = "Interpolation"
  p.X.Label.Text = "X"
  p.Y.Label.Text = "Y"

  steps := int((b - a) / h)
  f_p := make(plotter.XYs, steps)
  s_p := make(plotter.XYs, steps)
  x := a
  for step := 0; step < steps; step++ {
    f_p[step].X = x
    f_p[step].Y = f(x)
    s_p[step].X = x
    s_p[step].Y = ff(x)

    x += h
  }
  err := plotutil.AddLinePoints(p,
    "first derivative", f_p,
    "second derivative", s_p,
  )
  if err != nil {
    panic(err)
  }

  // Scatter
  scatter_data := make(plotter.XYs, len(points))
  for i := 0; i < len(points); i++ {
    scatter_data[i].X = points[i].x
    scatter_data[i].Y = points[i].y
  }
  s, err := plotter.NewScatter(scatter_data)
  if err != nil {
    panic(err)
  }
  s.GlyphStyle.Radius = 10
  p.Add(s)
  p.Legend.Add("Points", s)
  // Save the plot to a PNG file.
  if err := p.Save(2000, 2000, path); err != nil {
```

```
    panic(err)
  }
}

func main() {
  if len(os.Args) < 2 {
    panic("Аргументов должно быть два")
  }
  inputFile := os.Args[1]
  outputFile := os.Args[2]

  points := readFromFile(inputFile)
  fmt.Println(points)
  f := firstDerivative(points)
  fmt.Println("f'", f(0.2))

  ff := secondDerivative(points)
  fmt.Println("f\"", f(0.2))

  genPlot(outputFile, f, ff, points, points[0].x, points[len(points)-3].x, 0.01)
}
```

Вывод программы

f' 1.223500000000001

f" 1.223500000000001

Задание 3.5

Вычислить определенный интеграл, методами прямоугольников, Трапеций, Симпсона с шагами 0.5 и 0.25. Оценить погрешность вычислений, используя Метод Рунге-Ромберга.

Вариант:

$$f(x)=\frac{x}{(3x+4)^2}$$

Листинг

main.go:

```
package main

import (
  "fmt"
  "math"
)

type fn func(x float64) float64

func rectangleMethod(xs, xe, h float64, f fn) float64 {
  var res float64
  for x := xs + h; x <= xe; x += h {
    res += h * f(x-h/2)
  }
  return res
}

func trapezeMethod(xs, xe, h float64, f fn) float64 {
  var res float64
  for x := xs + h; x <= xe; x += h {
    res += h * (f(x) + f(x-h))
  }
  return res / 2
}

func simpsonMethod(xs, xe, h float64, f fn) float64 {
  var res float64
  h *= 2
  for x := xs + h; x <= xe; x += h {
    res += h * (f(x) + 4*f(x-h/2) + f(x-h))
  }
  return res / 6
}

func rungeRombergMethod(fh1, fh2, h1, h2 float64) float64 {
```

```
  k := h1 / h2
  return fh2 + (fh2-fh1)/(math.Pow(k, 2)-1)
}

func main() {
  var x0 float64 = -1
  var xk float64 = 1
  h1 := 0.5
  h2 := 0.25
  f := func(x float64) float64 {
    return x / math.Pow(3*x+4, 2)
  }

  {
    f1 := rectangleMethod(x0, xk, h1, f)
    f2 := rectangleMethod(x0, xk, h2, f)
    fmt.Println("Метод прямоугольников:", "h1: ", f1, "h2 ", f2)
    fmt.Println("Рунге-Ромберг-Ричардсон: ", rungeRombergMethod(f1, f2, h1, h2))

  }
  {
    f1 := trapezeMethod(x0, xk, h1, f)
    f2 := trapezeMethod(x0, xk, h2, f)
    fmt.Println("Метод трапеций:", "h1: ", f1, "h2 ", f2)
    fmt.Println("Рунге-Ромберг-Ричардсон: ", rungeRombergMethod(f1, f2, h1, h2))
  }
  {
    f1 := simpsonMethod(x0, xk, h1, f)
    f2 := simpsonMethod(x0, xk, h2, f)
    fmt.Println("Метод Симпсона:", "h1: ", f1, "h2 ", f2)
    fmt.Println("Рунге-Ромберг-Ричардсон: ", rungeRombergMethod(f1, f2, h1, h2))
  }
}
```

Вывод программы

Метод прямоугольников: h1: -0.1191431329134778 h2 -0.14931195938119501

Рунге-Ромберг-Ричардсон: -0.15936823487043408

Метод трапеций: h1: -0.27663349637375617 h2 -0.19788831464361697

Рунге-Ромберг-Ричардсон: -0.17163992073357057

Метод Симпсона: h1: -0.20557935570922584 h2 -0.1716399207335706

Рунге-Ромберг-Ричардсон: -0.16032677574168552