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»

Лабораторная работа № 2

Тема Билинейная интерполяция

Студент Кононенко Сергей

Группа ИУ7-43Б

Оценка (баллы) _____

Преподаватель Градов Владимир Михайлович

Москва — 2020 г.

1. Задание

1. Задана матрица значений функции вида x, y, $f(x, y)$. С помощью интерполяции, используя метод полинома Ньютона, найти приближённое значение функции от введённых x_0, y_0.
Ввод: значения x_0, y_0, степени полиномов n_x, n_y.
Вывод: значение функции от x_0, y_0.

2. Описание алгоритма

Билинейная интерполяция основывается на линейной интерполяции.

Для двух точек:

$$y(x_i, x_j) = \frac{y_i - y_j}{x_i - x_j}$$

Для трёх точек:

$$y(x_i, x_j, x_k) = \frac{y(x_i, x_j) - y(x_j, x_k)}{x_i - x_k}$$

Соответственно для n точек можно записать следующее выражение:

$$y(x_i, x_j, ..., x_n) = \frac{y(x_i, x_j, ..., x_{n-1}) - y(x_j, ..., x_n)}{x_i - x_n}$$

Отсюда искомый полином будет равен:

$$\mathcal{P}_n(x) = y_0 + (x - x_0)y(x_0, x_1) + (x - x_0)y(x_0, x_1)y(x_0, x_1, x_3) + ... + (x - x_0)...(x - x_n)y(x_0, x_1, ..., x_n)$$

Что эквивалентно: $\mathcal{P}_n(x) = \sum a_k x^k$

Первым шагом линейно интерполируется значение вспомогательных точек R_1 и R_2 вдоль оси абсцисс, где

$$R_1 = (x, y_1)$$

$$R_2 = (x, y_2)$$

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

Теперь проводится линейная интерполяция между вспомогательными точками R_1 и R_2. Это и есть интерполирующее значение функции $f(x, y)$

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

Таким образом, общая формула имеет вид

$$f(x, y) = \sum_{i=0}^{1} \sum_{j=0}^{1} a_{ij} x^i y^j = a_{00} + a_{10}x + a_{01}y + a_{11}xy$$

3. Код программы

Основной пакет приложения.

```
package main

import (
        "fmt"
        "os"

        "./interp"
)

func main() {
        if len(os.Args) <= 1 {
                fmt.Printf("USAGE: lab_02 <datafile>\n")
                os.Exit(1)
        }

        f, err := os.Open(os.Args[1])
        if err != nil {
                fmt.Println("Error:", err)
                os.Exit(1)
        }

        dm := interp.ReadDots(f)
        fmt.Printf("Table loaded from file:\n\n")
        dm.PrintMatrix()

        fmt.Printf("\nEnter X, Y value and X, Y polynom degrees: ")
        d, nx, ny := interp.ReadFuncData()

        Z := interp.BiInterpolation(dm, d, nx, ny)
        fmt.Printf("\nFunction value in (%5.2f;%5.2f) dot is %5.4f\n\n", d.X, d.Y, Z)
}
```

Пакет обработки интерполяции.

1. Обработка точек на плоскости.

```
package interp

import "fmt"

// Dot type used to represent cartesian dots.
type Dot struct {
        X float64
        Y float64
}
```

```
// DotSet type used to represent amount of cartesian dots.
type DotSet []Dot

// DotMatrix type used to represent matrix of cartesian dots.
type DotMatrix []DotSet

// PrintMatrix used to print X coords in DotMatrix.
func (dm DotMatrix) PrintMatrix() {
        for i := 0; i < len(dm); i++ {
                for j := 0; j < len(dm); j++ {
                        fmt.Printf("%8.2f ", dm[i][j].X)
                }
                fmt.Printf("\n")
        }
}

func (ds DotSet) getPos(d Dot) int {
        var pos int

        if d.X < ds[0].X {
                pos = 0
        } else if d.X > ds[len(ds)-1].X {
                pos = len(ds) - 1
        } else {
                for i := 1; i < len(ds)-2; i++ {
                        if d.X > ds[i-1].X && d.X <= ds[i].X {
                                pos = i - 1
                        }
                }
        }

        return pos
}
```

2. Реализация интерполяции.

```
package interp

import (
        "fmt"
        "io"
)

// FTable used to represent table of float numbers.
type FTable [][]float64

// BiInterpolation used to find value of X,Y Dot coordinate
```

```
// using Newton polynom.
func BiInterpolation(dm DotMatrix, d Dot, nx, ny int) float64 {
        xcol, yrow, zdm := splitData(dm)
        masterSet := make(DotSet, 0)

        baseX, startX, endX := getDimBase(xcol, d, nx)
        d.X, d.Y = d.Y, d.X
        baseY, startY, endY := getDimBase(yrow, d, ny)

        ftb := make(DotMatrix, nx+1)
        for i := range ftb {
                ftb[i] = make(DotSet, ny+1)
        }

        k := 0
        for i := startX; i < endX+1; i++ {
                l := 0
                for j := startY; j < endY+1; j++ {
                        ftb[k][l] = zdm[i][j]
                        l++
                }
                k++
        }

        for i := 0; i < len(ftb); i++ {
                for j := 0; j < len(baseY); j++ {
                        ftb[i][j].Y = baseY[j].X
                        ftb[i][j].X, ftb[i][j].Y = ftb[i][j].Y, ftb[i][j].X
                }
                md := Dot{
                        X: baseX[i].X,
                        Y: Interpolation(ftb[i], d, ny),
                }
                masterSet = append(masterSet, md)
        }

        d.X, d.Y = d.Y, d.X

        return Interpolation(masterSet, d, nx)
}

// Interpolation used to find value of X Dot coordinate
// using Newton polynom.
func Interpolation(ds DotSet, d Dot, n int) float64 {
        tb := makeTable(ds, d, n)
        p := tb[1][0]
        var c float64
```

```
        for i := 2; i < len(tb); i++ {
                c = 1
                for j := 0; j < i-1; j++ {
                        c *= d.X - tb[0][j]
                }
                c *= tb[i][0]
                p += c

        }

        return p
}

// ReadFuncData used to read function X coordinate and polynom degree.
func ReadFuncData() (Dot, int, int) {
        var (
                d      Dot
                nx, ny int
        )

        fmt.Scan(&d.X, &d.Y, &nx, &ny)

        return d, nx, ny
}

// ReadDots used to read Dot objects to DotSet object from file.
func ReadDots(f io.Reader) DotMatrix {
        var (
                ds   int
                data DotMatrix
        )

        fmt.Fscanln(f, &ds)

        data = make(DotMatrix, ds)
        for i := range data {
                data[i] = make(DotSet, ds)
        }

        for i := 0; i < ds; i++ {
                for j := 0; j < ds; j++ {
                        fmt.Fscan(f, &data[i][j].X)
                }
        }

        return data
}
```

```
func getDimBase(ds DotSet, d Dot, n int) (DotSet, int, int) {
	base := getBase(ds, d, n)
	start, end := 0, 0

	for i, el := range ds {
		if el == base[0] {
			start = i
			end = start + n
			break
		}
	}

	return base, start, end
}

func makeTable(ds DotSet, d Dot, n int) FTable {
	base := getBase(ds, d, n)
	baselen := len(base)
	tb := make(FTable, n+2)
	tblen := len(tb)
	for i := range tb {
		tb[i] = make([]float64, baselen)
	}

	for i := range tb[0] {
		tb[0][i] = base[i].X
		tb[1][i] = base[i].Y
	}

	for i := 2; i < tblen; i++ {
		k := i - 2
		for j := 0; j < baselen-i+1; j++ {
			tb[i][j] = (tb[i-1][j] - tb[i-1][j+1]) /
				(tb[0][j] - tb[0][j+k+1])
		}
	}

	return tb
}

func getBase(ds DotSet, d Dot, n int) DotSet {
	base := DotSet{}
	pos := ds.getPos(d)

	if pos <= n/2 {
		for i := 0; i < n+1; i++ {
			base = append(base, ds[i])
		}
```

```
	} else if len(ds)-pos-1 <= n/2 {
		for i := len(ds) - n - 1; i < len(ds); i++ {
			base = append(base, ds[i])
		}
	} else {
		lb := n / 2
		rb := n - lb + 1
		if pos+rb > len(ds)-1 {
			rb--
			lb++
		}
		if pos-lb < 0 {
			rb++
			lb--
		}

		for i := pos - lb; i < pos+rb; i++ {
			base = append(base, ds[i])
		}
	}

	return base
}

func splitData(dm DotMatrix) (DotSet, DotSet, DotMatrix) {
	xcol := make(DotSet, 0)
	for i := 1; i < len(dm); i++ {
		xcol = append(xcol, dm[i][0])
	}

	yrow := dm[0][1:]

	zdm := make(DotMatrix, len(dm)-1)
	for i := range zdm {
		zdm[i] = make(DotSet, len(dm)-1)
	}

	for i := 1; i < len(dm); i++ {
		for j := 1; j < len(dm); j++ {
			zdm[i-1][j-1] = dm[i][j]
		}
	}

	return xcol, yrow, zdm
}
```

3. Генерация тестовых данных.

```
package main

import (
        "fmt"
        "os"
        "strconv"

        "../interp"
)

func main() {
        if len(os.Args) < 5 {
                fmt.Printf("USAGE: data <start> <end> <step> <datafile>\n")
                os.Exit(1)
        }

        f, err := os.Create(os.Args[4])
        if err != nil {
                fmt.Println("Error:", err)
                os.Exit(1)
        }

        start, _ := strconv.ParseFloat(os.Args[1], 64)
        end, _ := strconv.ParseFloat(os.Args[2], 64)
        step, _ := strconv.ParseFloat(os.Args[3], 64)

        data := genData(start, end, step)
        ds := len(data)

        f.WriteString(fmt.Sprintf("%d\n", ds))
        for i := 0; i < ds; i++ {
                for j := 0; j < ds; j++ {
                        f.WriteString(fmt.Sprintf("%8.2f", data[i][j]))
                }
                f.WriteString("\n")
        }

        f.Close()
}

func genData(start, end, step float64) interp.FTable {
        size := int((end + start) / step)
        data := make(interp.FTable, size)
        for i := range data {
                data[i] = make([]float64, size)
        }
```

```
        for i := 1; i < size; i++ {
                data[0][i] = start + step*float64(i-1)
                data[i][0] = start + step*float64(i-1)
        }

        for i := 1; i < size; i++ {
                for j := 1; j < size; j++ {
                        data[i][j] = mathFunc(data[0][i], data[j][0])
                }
        }

        return data
}

func mathFunc(x, y float64) float64 {
        return x*x - y*y*y
}
```