Московский Авиационный Институт $_h$
(национальный исследовательский университет)

Факультет Компьютерных наук и прикладной математики

Кафедра Вычислительной математики и программирования

Лабораторные работы

по дисциплине

«Численные методы»

III курс, VI семестр

Студент: Синюков А.С.

Группа: М8О-306Б-21

Руководитель: Ревизников Д. Л.

Москва, 2024

Оглавление

Лабораторная работа №1

Задание:

1.1. Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.

1.2. Реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.

1.3. Реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.

1.4. Реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.

1.5. Реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.

Код:

Программа реализована на языке программирования Go.

LU.go:

```
func GetLU(m [][]float64, mSize int) ([][]float64, [][]float64, [][]float64) {
    var L [][]float64
    var U [][]float64
    var P [][]float64

    for i := 0; i < mSize; i++ {
        L = append(L, make([]float64, mSize))
        U = append(U, make([]float64, mSize))
        P = append(P, make([]float64, mSize))
    }

    for i := 0; i < mSize; i++ {
        L[i][i] = 1
        P[i][i] = 1
    }

    for i := 0; i < mSize; i++ {
        for j := 0; j < mSize; j++ {
            U[i][j] = m[i][j]
        }
    }

    for i := 0; i < mSize; i++ {
        kMax := i
        maxx := math.Inf(-1)
        for k := i; k < mSize; k++ {
            if U[i][k] > maxx {
                maxx = U[i][k]
                kMax = k
            }
        }
        for k := 0; k < mSize; k++ {
            temp := U[i][k]
            U[i][k] = U[kMax][k]
            U[kMax][k] = temp
        }
        for k := 0; k < mSize; k++ {
            temp := L[i][k]
            L[i][k] = L[kMax][k]
            L[kMax][k] = temp
        }
        for k := 0; k < mSize; k++ {
```

```
            temp := L[k][i]
            L[k][i] = L[k][kMax]
            L[k][kMax] = temp
        }
        for k := 0; k < mSize; k++ {
            temp := P[k][i]
            P[k][i] = P[k][kMax]
            P[k][kMax] = temp
        }
        for j := i + 1; j < mSize; j++ {
            l := U[j][i] / U[i][i]
            L[j][i] = l
            for k := 0; k < mSize; k++ {
                U[j][k] = U[j][k] - (U[i][k] * L[j][i])
            }
        }
    }

    return L, U, P
}

func DetLU(m [][]float64, mSize int) float64 {
    _, U, P := GetLU(m, mSize)
    sign := 1
    for i := 0; i < mSize; i++ {
        if P[i][i] != 1 {
            sign *= -1
        }
    }
    res := 1.0
    for i := 0; i < mSize; i++ {
        res *= U[i][i]
    }
    return res * float64(sign)
}

func SolveLU(m [][]float64, b []float64, mSize int) []float64 {
    L, U, P := GetLU(m, mSize)
    return SolveWithLU(L, U, P, b, mSize)
}

func SolveWithLU(L [][]float64, U [][]float64, P [][]float64, b
[]float64, mSize int) []float64 {
    y := make([]float64, mSize)
    b = iterations.MatrixVectorMult(P, b)
    for i := 0; i < mSize; i++ {
        temp := 0.0
        for j := 0; j < i; j++ {
            temp += L[i][j] * y[j]
        }
```

```
        y[i] = (b[i] - temp) / L[i][i]
    }
    x := make([]float64, mSize)
    for i := mSize - 1; i >= 0; i-- {
        temp := 0.0
        for j := mSize - 1; j >= i; j-- {
            temp += U[i][j] * x[j]
        }
        x[i] = (y[i] - temp) / U[i][i]
    }
    return x
}

func InvertLU(m [][]float64, mSize int) [][]float64 {
    e := make([]float64, mSize)
    e[0] = 1
    var res [][]float64
    for i := 0; i < mSize; i++ {
        res = append(res, make([]float64, mSize))
    }
    L, U, P := GetLU(m, mSize)
    for i := 1; i <= mSize; i++ {
        st := SolveWithLU(L, U, P, e, mSize)
        for j := 0; j < mSize; j++ {
            res[j][i-1] = st[j]
        }
        e[i-1] = 0
        if i != mSize {
            e[i] = 1
        }
    }

    return res
}
```

tridiagonal.go:

```
func MatrixToTridiagonal(m [][]float64, mSize int) ([]float64, []float64,
[]float64) {
    A := make([]float64, mSize)
    B := make([]float64, mSize)
    C := make([]float64, mSize)
    for i := 0; i < mSize; i++ {
        if i > 0 {
            A[i] = m[i][i-1]
        }
        B[i] = m[i][i]
        if i < mSize-1 {
            C[i] = m[i][i+1]
```

```
        }
    }
    return A, B, C
}

func SolveTridiagonal(m [][]float64, d []float64, mSize int) []float64 {
    a, b, c := MatrixToTridiagonal(m, mSize)

    p := make([]float64, mSize)
    q := make([]float64, mSize)

    p[0] = -c[0] / b[0]
    q[0] = d[0] / b[0]

    for i := 1; i < mSize; i++ {
        p[i] = -c[i] / (b[i] + a[i]*p[i-1])
        q[i] = (d[i] - a[i]*q[i-1]) / (b[i] + a[i]*p[i-1])
    }

    x := make([]float64, mSize)
    x[mSize-1] = q[mSize-1]

    for i := mSize - 2; i >= 0; i-- {
        x[i] = p[i]*x[i+1] + q[i]
    }

    return x
}
```

iterations.go:

```
func MatrixNormal(m [][]float64) float64 {
    res := math.Inf(-1)
    for i := 0; i < len(m); i++ {
        s := 0.0
        for j := 0; j < len(m); j++ {
            s += math.Abs(m[i][j])
        }
        res = math.Max(res, s)
    }
    return res
}

func VectorNormal(v []float64) float64 {
    res := math.Inf(-1)
    for i := 0; i < len(v); i++ {
        res = math.Max(res, math.Abs(v[i]))
    }
    return res
```

```
}

func Jakobi(a [][]float64, b []float64, mSize int) ([][]float64,
[]float64) {
    var alpha [][]float64
    for i := 0; i < mSize; i++ {
        alpha = append(alpha, make([]float64, mSize))
    }
    for i := 0; i < mSize; i++ {
        for j := 0; j < mSize; j++ {
            if i != j {
                alpha[i][j] = -a[i][j] / a[i][i]
            }
        }
    }

    beta := make([]float64, mSize)
    for i := 0; i < mSize; i++ {
        beta[i] = b[i] / a[i][i]
    }
    return alpha, beta
}

func MatrixVectorMult(a [][]float64, b []float64) []float64 {
    //pls matching matrix & vector
    mSize := len(b)
    c := make([]float64, mSize)
    for i := 0; i < mSize; i++ {
        for j := 0; j < mSize; j++ {
            c[i] += b[j] * a[i][j]
        }
    }
    return c
}

func VectorSum(a []float64, b []float64) []float64 {
    vSize := len(b)
    c := make([]float64, vSize)
    for i := 0; i < vSize; i++ {
        c[i] = a[i] + b[i]
    }
    return c
}

func VectorSubstr(a []float64, b []float64) []float64 {
    vSize := len(b)
    for i := 0; i < vSize; i++ {
        b[i] = -b[i]
    }
    return VectorSum(a, b)
```

```
}

func SolveSimpleIt(m [][]float64, b []float64, eps float64, mSize int)
([]float64, int) {
    alpha, beta := Jakobi(m, b, mSize)
    currEps := math.Inf(1)
    aNorm := MatrixNormal(alpha)
    count := 0
    x := VectorSum(beta, make([]float64, mSize))
    for eps < currEps {
        c := MatrixVectorMult(alpha, x)
        newX := VectorSum(beta, c)
        currEps = (aNorm / (1 - aNorm)) * VectorNormal(VectorSubstr(newX,
x))
        x = newX
        count++
    }
    return x, count
}

func SolveZeidel(m [][]float64, b []float64, eps float64, mSize int)
([]float64, int) {
    alpha, beta := Jakobi(m, b, mSize)

    var c [][]float64
    for i := 0; i < mSize; i++ {
        c = append(c, make([]float64, mSize))
    }
    for i := 0; i < mSize; i++ {
        for j := 0; j < mSize; j++ {
            c[i][j] = alpha[i][j]
        }
    }

    currEps := math.Inf(1)
    aNorm := MatrixNormal(alpha)
    cNorm := MatrixNormal(c)
    count := 0
    x := VectorSum(beta, make([]float64, mSize))
    for eps < currEps {
        newX := VectorSum(beta, make([]float64, mSize))
        for i := 0; i < mSize; i++ {
            for j := 0; j < i; j++ {
                newX[i] += newX[j] * alpha[i][j]
            }
            for j := i; j < mSize; j++ {
                newX[i] += x[j] * alpha[i][j]
            }
        }
        currEps = (cNorm / (1 - aNorm)) * VectorNormal(VectorSubstr(newX,
```

```
x))
        x = newX
        count++
    }
    return x, count
}
```

rotations.go:

```
func MatrixSquares(m [][]float64) float64 {
    res := 0.0
    for i := 0; i < len(m); i++ {
        for j := i + 1; j < len(m); j++ {
            res += m[i][j] * m[i][j]
        }
    }
    return math.Sqrt(res)
}

func GetMaxNoDiag(m [][]float64, mSize int) (int, int) {
    iMax := 0
    jMax := 0
    maxElem := math.Inf(-1)
    for i := 0; i < mSize; i++ {
        for j := 0; j < mSize; j++ {
            if i != j {
                if math.Abs(m[i][j]) > maxElem {
                    iMax = i
                    jMax = j
                    maxElem = math.Abs(m[i][j])
                }
            }
        }
    }
    return iMax, jMax
}

func GetRotationMatrix(mSize int, phi float64, i int, j int) [][]float64
{
    var u [][]float64
    for k := 0; k < mSize; k++ {
        u = append(u, make([]float64, mSize))
    }
    for k := 0; k < mSize; k++ {
        u[k][k] = 1
    }
    u[i][i] = math.Cos(phi)
    u[j][j] = math.Cos(phi)
    u[i][j] = -math.Sin(phi)
```

```
    u[j][i] = math.Sin(phi)
    return u
}

func MatrixMult(a [][]float64, b [][]float64, mSize int) [][]float64 {
    var c [][]float64
    for i := 0; i < mSize; i++ {
        c = append(c, make([]float64, mSize))
    }

    for i := 0; i < mSize; i++ {
        for j := 0; j < mSize; j++ {
            for k := 0; k < mSize; k++ {
                c[i][j] += a[i][k] * b[k][j]
            }
        }
    }
    return c
}

func MatrixTranspose(m [][]float64, mSize int) [][]float64 {
    var c [][]float64
    for i := 0; i < mSize; i++ {
        c = append(c, make([]float64, mSize))
    }

    for i := 0; i < mSize; i++ {
        for j := 0; j < mSize; j++ {
            c[i][j] = m[j][i]
        }
    }
    return c
}

func Rotation(m [][]float64, mSize int, eps float64) ([][]float64, [][]float64, int) {
    var mNew [][]float64
    for i := 0; i < mSize; i++ {
        mNew = append(mNew, m[i])
    }
    var u [][]float64
    for i := 0; i < mSize; i++ {
        u = append(u, make([]float64, mSize))
    }
    for i := 0; i < mSize; i++ {
        u[i][i] = 1
    }
    count := 0
    var phi float64
    for MatrixSquares(mNew) > eps {
```

```
        i, j := GetMaxNoDiag(mNew, mSize)
        if mNew[i][i] == mNew[j][j] {
            phi = math.Pi / 4
        } else {
            phi = 0.5 * math.Atan2(2*mNew[i][j], (mNew[i][i]-mNew[j][j]))
        }
        uNew := GetRotationMatrix(mSize, phi, i, j)
        u = MatrixMult(u, uNew, mSize)
        mNew = MatrixMult(MatrixMult(MatrixTranspose(uNew, mSize), mNew,
mSize), uNew, mSize)
        count++
    }
    return mNew, u, count
}
```

QR.go:

```
func VectorVectorMatrix(a []float64, b []float64) [][]float64 {
    var c [][]float64
    for i := 0; i < len(a); i++ {
        c = append(c, make([]float64, len(a)))
    }
    for i := 0; i < len(a); i++ {
        for j := 0; j < len(a); j++ {
            c[i][j] = a[i] * b[j]
        }
    }
    return c
}

func MatrixMult(a [][]float64, b [][]float64, mSize int) [][]float64 {
    var c [][]float64
    for i := 0; i < mSize; i++ {
        c = append(c, make([]float64, mSize))
    }

    for i := 0; i < mSize; i++ {
        for j := 0; j < mSize; j++ {
            for k := 0; k < mSize; k++ {
                c[i][j] += a[i][k] * b[k][j]
            }
        }
    }
    return c
}

func MatrixTranspose(m [][]float64, mSize int) [][]float64 {
    var c [][]float64
    for i := 0; i < mSize; i++ {
```

```
        c = append(c, make([]float64, mSize))
    }

    for i := 0; i < mSize; i++ {
        for j := 0; j < mSize; j++ {
            c[i][j] = m[j][i]
        }
    }
    return c
}

func VectorVectorNumber(a []float64, b []float64) float64 {
    c := 0.0
    for i := 0; i < len(a); i++ {
        c += a[i] * b[i]
    }
    return c
}

func MatrixNumberMult(a [][]float64, n float64) [][]float64 {
    var c [][]float64
    for i := 0; i < len(a); i++ {
        c = append(c, make([]float64, len(a)))
    }
    for i := 0; i < len(a); i++ {
        for j := 0; j < len(a); j++ {
            c[i][j] = a[i][j] * n
        }
    }
    return c
}

func MatrixSum(a [][]float64, b [][]float64) [][]float64 {
    var c [][]float64
    for i := 0; i < len(a); i++ {
        c = append(c, make([]float64, len(a)))
    }
    for i := 0; i < len(a); i++ {
        for j := 0; j < len(a); j++ {
            c[i][j] = a[i][j] + b[i][j]
        }
    }
    return c
}

func MatrixSubstr(a [][]float64, b [][]float64) [][]float64 {
    for i := 0; i < len(a); i++ {
        for j := 0; j < len(a); j++ {
            b[i][j] = -b[i][j]
        }
```

```
    }
    return MatrixSum(a, b)
}

func VectorSum(a []float64, b []float64) []float64 {
    vSize := len(b)
    c := make([]float64, vSize)
    for i := 0; i < vSize; i++ {
        c[i] = a[i] + b[i]
    }
    return c
}

func VectorSubstr(a []float64, b []float64) []float64 {
    vSize := len(b)
    for i := 0; i < vSize; i++ {
        b[i] = -b[i]
    }
    return VectorSum(a, b)
}

func VectorSquares(v []float64) float64 {
    res := 0.0
    for i := 0; i < len(v); i++ {
        res += v[i] * v[i]
    }
    return math.Sqrt(res)
}

func Sign(x float64) int {
    switch {
    case x > 0:
        return 1
    case x == 0:
        return 0
    default:
        return -1
    }
}

func GetHouseholderMatrix(mSize int, v []float64, i int) [][]float64 {
    vNew := VectorSum(v, make([]float64, mSize))
    vNew[i] += float64(Sign(v[i])) * VectorSquares(v)
    var e [][]float64
    for k := 0; k < mSize; k++ {
        e = append(e, make([]float64, mSize))
    }
    for k := 0; k < mSize; k++ {
        e[k][k] = 1
    }
```

```
    return MatrixSubstr(e, MatrixNumberMult(VectorVectorMatrix(vNew,
vNew), 2/(VectorVectorNumber(vNew, vNew))))
}

func GetQR(m [][]float64, mSize int) ([][]float64, [][]float64) {
    var Q [][]float64
    for i := 0; i < mSize; i++ {
        Q = append(Q, make([]float64, mSize))
    }
    for i := 0; i < mSize; i++ {
        Q[i][i] = 1.0
    }
    var R [][]float64
    for i := 0; i < mSize; i++ {
        R = append(R, m[i])
    }
    for i := 0; i < mSize-1; i++ {
        b := make([]float64, mSize)
        for j := i; j < mSize; j++ {
            b[j] = R[j][i]
        }
        H := GetHouseholderMatrix(mSize, b, i)
        Q = MatrixMult(Q, H, mSize)
        R = MatrixMult(H, R, mSize)
    }
    return Q, R
}

func ComplexSolve(a11, a12, a21, a22, eps float64) (complex128,
complex128, bool) {
    a := 1.0
    b := -a11 - a22
    c := a11*a22 - a12*a21
    d := b*b - 4*a*c
    if d > eps {
        return complex(0, 0), complex(0, 0), false
    } else {
        return (complex(-b, 0) + complex(0, math.Sqrt(-d))) /
complex(2*a, 0),
            (complex(-b, 0) - complex(0, math.Sqrt(-d))) / complex(2*a,
0), true
    }

}

func QRAlgo(m [][]float64, mSize int, eps float64) ([]complex128, int) {
    var mNew [][]float64
    for i := 0; i < mSize; i++ {
        mNew = append(mNew, m[i])
    }
```

```
    count := 0
    res := make([]complex128, mSize)
    for true {
        count++
        Q, R := GetQR(mNew, mSize)
        mNew = MatrixMult(R, Q, mSize)
        br := true
        i := 0
        for i < mSize {
            if i < mSize-1 && math.Abs(mNew[i+1][i]) > eps {
                e1, e2, f := ComplexSolve(mNew[i][i], mNew[i][i+1],
mNew[i+1][i], mNew[i+1][i+1], eps)
                if f {
                    res[i] = e1
                    res[i+1] = e2
                    i++
                } else {
                    br = false
                }
            } else {
                res[i] = complex(mNew[i][i], 0)
            }
            i++
        }
        if br {
            break
        }
    }
    return res, count
}
```

Пример работы:

1.1)

```
C:\Users\sinyu\Desktop\numeric\lab1>go run main.go <1.txt
Select lab 1.X:
1: LU-decomposition
2: Tridiagonal matrix algo
3: Iterations/Zeidel algo
4: Rotations method
5: QR-decomposition
1.1-----LU-----
Size:
Matrix:
b:
L:
1.00 0.00 0.00 0.00
0.62 1.00 0.00 0.00
1.00 1.44 1.00 0.00
1.00 0.56 0.69 1.00
U:
8.00 8.00 -5.00 8.00
0.00 -9.00 -2.88 -7.00
0.00 0.00 18.15 -5.89
0.00 0.00 0.00 5.98
P:
1 0 0 0
0 0 1 0
0 1 0 0
0 0 0 1
det:  -7810.000000000001
inv:
0.04 0.04 0.04 0.01
0.16 0.08 -0.15 -0.15
-0.04 0.02 -0.06 0.05
-0.10 -0.12 0.07 0.17
solve:
1.408451
17.158771
-5.752881
-20.537772
```

1.2)

```
C:\Users\sinyu\Desktop\numeric\lab1>go run main.go <2.txt
Select lab 1.X:
1: LU-decomposition
2: Tridiagonal matrix algo
3: Iterations/Zeidel algo
4: Rotations method
5: QR-decomposition
1.2-----TRIDIAGONAL-----
Size:
Matrix:
b:
A: 0.00 -1.00 -9.00 -1.00 9.00
B: -6.00 13.00 -15.00 -7.00 -18.00
C: 5.00 6.00 -4.00 1.00 0.00
solve:
-1.000000
9.000000
-3.000000
-6.000000
2.000000
```

1.3)

```
C:\Users\sinyu\Desktop\numeric\lab1>go run main.go <3.txt
Select lab 1.X:
1: LU-decomposition
2: Tridiagonal matrix algo
3: Iterations/Zeidel algo
4: Rotations method
5: QR-decomposition
1.3-----ITERATIONS-----
Size:
Matrix:
b:
eps:
iterations solve:
-2.000000
2.000000
-4.000000
1.000000
count:
65

Zeidel solve:
-2.000000
2.000000
-4.000000
1.000000
count:
42
```

1.4)

```
C:\Users\sinyu\Desktop\numeric\lab1>go run main.go <4.txt
Select lab 1.X:
1: LU-decomposition
2: Tridiagonal matrix algo
3: Iterations/Zeidel algo
4: Rotations method
5: QR-decomposition
1.4-----ROTATION-----
Size:
Matrix:
eps:
eigenvectors:
(0.7803 -0.3643 -0.5084 )
(0.6250 0.4232 0.6560 )
(-0.0238 -0.8296 0.5579 )

eigenvalues:
9.006615
-1.230143
-5.776472

Check 1:
(7.0275 -3.2815 -4.5787 )
(7.0278 -3.2813 -4.5784 )
Check 2:
(-0.7688 -0.5206 -0.8069 )
(-0.7684 -0.5208 -0.8072 )
Check 3:
(0.1377 4.7919 -3.2228 )
(0.1377 4.7919 -3.2228 )

count:
6
```

1.5)

```
C:\Users\sinyu\Desktop\numeric\lab1>go run main.go <5.txt
Select lab 1.X:
1: LU-decomposition
2: Tridiagonal matrix algo
3: Iterations/Zeidel algo
4: Rotations method
5: QR-decomposition
1.5-----QR-----
Size:
Matrix:
eps:
Q:
-0.6882 0.5311 -0.4942
-0.6882 -0.2626 0.6763
0.2294 0.8056 0.5462

R:
8.7178 -8.2590 5.2766
0.0000 -8.8198 2.4287
-0.0000 0.0000 3.3555

eigenvalues:
8.3211 -6.2676 4.9465
count:
33
```

Лабораторная работа №2

Задание:

2.1. Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

2.2. Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

Код:

Программа реализована на языке программирования Go.

equations.go:

```
func SimpleIterationsMethod(phi, dphi func(float64) float64, a, b, eps
float64) (float64, int) {
    q := math.Min(math.Abs(dphi(a)), math.Abs(dphi(b)))
    x := b
    dx := math.Inf(+1)
    count := 0
    for eps < dx*(q/(1-q)) {
        xNew := phi(x)
        dx = math.Abs(xNew - x)
        x = xNew
        count++
    }
    return x, count
}

func NewtonMethod(f, df, d2f func(float64) float64, eps, x0 float64)
(float64, int) {
    if f(x0)*d2f(x0) <= 0 {
        return 0.0, -1
    }
    x := x0
    dx := math.Inf(+1)
    count := 0
    for eps < dx {
        xNew := x - f(x)/df(x)
        dx = math.Abs(xNew - x)
        x = xNew
        count++
    }
    return x, count
}
```

systems.go:

```
func GetQ(dphi [][]func([]float64) float64, a1, b1, a2, b2 float64)
float64 {
    x1 := (a1+b1)/2 + math.Abs(b1-a1)
    x2 := (a2+b2)/2 + math.Abs(b2-a2)
    var x []float64
    x = append(x, x1, x2)
    return math.Max(math.Abs(dphi[0][0](x))+math.Abs(dphi[0][1](x)),
math.Abs(dphi[1][0](x))+math.Abs(dphi[1][1](x)))
}
```

```
func VectorNormal(v []float64) float64 {
    res := math.Inf(-1)
    for i := 0; i < len(v); i++ {
        res = math.Max(res, math.Abs(v[i]))
    }
    return res
}

func VectorSum(a []float64, b []float64) []float64 {
    vSize := len(b)
    c := make([]float64, vSize)
    for i := 0; i < vSize; i++ {
        c[i] = a[i] + b[i]
    }
    return c
}

func VectorSubstr(a []float64, b []float64) []float64 {
    vSize := len(b)
    for i := 0; i < vSize; i++ {
        b[i] = -b[i]
    }
    return VectorSum(a, b)
}

func SimpleIterationsMethod(phi []func([]float64) float64, dphi []
[]func([]float64) float64, a1, b1, a2, b2, eps float64) ([]float64, int)
{
    q := GetQ(dphi, a1, b1, a2, b2)
    x := append(make([]float64, 0), a1, b1)
    dx := math.Inf(+1)
    count := 0
    for eps < dx*(q/(1-q)) {
        xNew := append(make([]float64, 0), phi[0](x), phi[1](x))
        dx = VectorNormal(VectorSubstr(xNew, x))
        x = xNew
        count++
    }
    return x, count
}

func GetMatrixJ(df [][]func([]float64) float64, x []float64) [][]float64
{
    var J [][]float64
    for i := 0; i < 2; i++ {
        J = append(J, append(make([]float64, 0), df[i][0](x), df[i][1]
(x)))
    }
    return J
```

```
}

func GetVectorB(f []func([]float64) float64, x []float64) []float64 {
    var b []float64
    for i := 0; i < 2; i++ {
        b = append(b, f[i](x))
    }
    return b
}

func NewtonMethod(f []func([]float64) float64, df [][]func([]float64)
float64, a1, b1, eps float64) ([]float64, int) {
    dx := math.Inf(+1)
    x := append(make([]float64, 0), a1, b1)
    count := 0
    for eps < dx {
        J := GetMatrixJ(df, x)
        b := GetVectorB(f, x)
        delta := LU.SolveLU(J, b, len(b))
        xNew := VectorSubstr(x, delta)
        dx = VectorNormal(VectorSubstr(xNew, x))
        x = xNew
        count++
    }
    return x, count
}
```

Пример работы:

2.1)

```
C:\Users\sinyu\Desktop\numeric\lab2>go run main.go
Select lab 2.X:
1: equation
2: system
1
2.1-----Equations-----
Simple Iterations:
answer:  1.69460512203764
count:  6

Newton:
answer:  1.6941445689789367
count:  4
```

2.2)

```
C:\Users\sinyu\Desktop\numeric\lab2>go run main.go
Select lab 2.X:
1: equation
2: system
2
2.2-----Systems-----
Simple Iterations:
x1 = 0.300142
x2 = 0.450020
count:  8

Newton:
x1 = 0.300146
x2 = 0.450019
count:  4
```

Лабораторная работа №3

Задание:

3.1. Используя таблицу значений Y_i функции y=f(x), вычисленных в точках X_i, i = 0, …,3 построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки $\{X_i, Y_i\}$. Вычислить значение погрешности интерполяции в точке X^*.

3.2. Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при $x=x_0$ и $x=x_4$. Вычислить значение функции в точке $x=X^*$.

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.

3.4. Вычислить первую и вторую производную от таблично заданной функции $y_i = f(x_i)$, i = 0, …,4 в точке $x=X^*$.

3.5. Вычислить определенный интеграл $$F = \int_{X_0}^{X_1} y\,dx,$$ методами прямоугольников, трапеций, Симпсона с шагами h1, h2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга.

Код:

Программа реализована на языке программирования Go.

interpolation.go:

```
func PolyMult(x1, x2 []float64) []float64 {
    res := make([]float64, len(x1)+len(x2))
    for i := 0; i < len(x1); i++ {
        for j := 0; j < len(x2); j++ {
            res[i+j] += x1[i] * x2[j]
        }
    }
    return res
}

func PolyPlus(x1, x2 []float64, n int) []float64 {
    res := make([]float64, n)
    for i := 0; i < n; i++ {
        if i >= len(x2) {
            res[i] = x1[i]
        } else if i >= len(x1) {
            res[i] = x2[i]
        } else {
            res[i] = x1[i] + x2[i]
        }
    }
    return res
}

func PolyScalarMult(x1 []float64, k float64, n int) []float64 {
    res := make([]float64, n)
    for i := 0; i < n; i++ {
        res[i] = x1[i] * k
    }
    return res
}

func PolynomLagrange(x, y []float64, n int) []float64 {
    var res []float64
    res = append(res, 0)
    for i := 0; i < n; i++ {
        li := make([]float64, 1)
        li[0] = 1
        for j := 0; j < n; j++ {
            if i == j {
                continue
            }
            xx := make([]float64, 2)
```

```
            xx[0] = -x[j]
            xx[1] = 1
            li = PolyMult(li, xx)
            li = PolyScalarMult(li, 1/(x[i]-x[j]), len(li))
        }
        res = PolyPlus(res, PolyScalarMult(li, y[i], len(li)),
int(math.Max(float64(len(li)), float64(len(res)))))
    }
    return res
}

func diff(x, y []float64, l, r int) float64 {
    if l+1 == r {
        return (y[l] - y[r]) / (x[l] - x[r])
    } else {
        return (diff(x, y, l, r-1) - diff(x, y, l+1, r)) / (x[l] - x[r])
    }
}

func PolynomNewton(x, y []float64, n int) []float64 {
    j := 1
    var res []float64
    res = append(res, y[0])
    var li []float64
    li = append(li, -x[0], 1)
    for i := 1; i < n; i++ {
        res = PolyPlus(res, PolyScalarMult(li, diff(x, y, 0, j),
len(li)), int(math.Max(float64(len(li)), float64(len(res)))))
        xx := make([]float64, 2)
        xx[0] = -x[i]
        xx[1] = 1
        li = PolyMult(li, xx)
        j++
    }
    return res
}
```

spline.go:

```
func Spline(x, f []float64, n int) ([]float64, []float64, []float64,
[]float64) {
    a := make([]float64, n)
    b := make([]float64, n)
    c := make([]float64, n)
    d := make([]float64, n)

    h := make([]float64, n)
    for i := 1; i < n; i++ {
        h[i] = x[i] - x[i-1]
```

```
    }

    ta := make([]float64, n-2)
    tb := make([]float64, n-2)
    tc := make([]float64, n-2)
    td := make([]float64, n-2)
    for i := 0; i < n-2; i++ {
        ta[i] = h[i+1]
        tb[i] = 2 * (h[i+1] + h[i+2])
        tc[i] = h[i+2]
        td[i] = 3 * ((f[i+2]-f[i+1])/h[i+2] - (f[i+1]-f[i])/h[i+1])
    }
    ta[0] = 0
    tc[n-3] = 0
    solvedC := tridiagonal.SolveTridiagonal(ta, tb, tc, td, n-2)

    for i := 1; i < n; i++ {
        a[i] = f[i-1]
        if i > 1 {
            c[i] = solvedC[i-2]
        }
    }
    c[1] = 0.0
    for i := 1; i < n-1; i++ {
        b[i] = (f[i]-f[i-1])/h[i] - h[i]*(c[i+1]+2*c[i])/3.0
        d[i] = (c[i+1] - c[i]) / (3 * h[i])
    }
    b[n-1] = ((f[n-1] - f[n-2]) / h[n-1]) - ((2.0 / 3.0) * h[n-1] * c[n-
1])
    d[n-1] = -c[n-1] / (3 * h[n-1])
    return a, b, c, d
}
```

LSM.go:

```
func MatrixTranspose(m [][]float64) [][]float64 {
    var c [][]float64
    for i := 0; i < len(m[0]); i++ {
        c = append(c, make([]float64, len(m)))
    }

    for i := 0; i < len(c); i++ {
        for j := 0; j < len(c[0]); j++ {
            c[i][j] = m[j][i]
        }
    }
    return c
}
```

```
func MatrixMult(a [][]float64, b [][]float64) [][]float64 {
    var c [][]float64
    for i := 0; i < len(a); i++ {
        c = append(c, make([]float64, len(b[0])))
    }

    for i := 0; i < len(a); i++ {
        for j := 0; j < len(b[0]); j++ {
            for k := 0; k < len(b); k++ {
                c[i][j] += a[i][k] * b[k][j]
            }
        }
    }
    return c
}

func MatrixVectorMult(a [][]float64, b []float64) []float64 {
    c := make([]float64, len(a))
    for i := 0; i < len(a); i++ {
        for j := 0; j < len(a[0]); j++ {
            c[i] += b[j] * a[i][j]
        }
    }
    return c
}

func A(x, y []float64, p, n int) []float64 {
    var a [][]float64
    for i := 0; i < n; i++ {
        a = append(a, make([]float64, p+1))
    }
    for i := 0; i < n; i++ {
        for j := 0; j < p+1; j++ {
            a[i][j] = math.Pow(x[i], float64(j))
        }
    }
    at := MatrixTranspose(a)
    r := MatrixVectorMult(at, y)
    ata := MatrixMult(at, a)
    return LU.SolveLU(ata, r, len(r))
}

func Error(x, y, a []float64) float64 {
    POLY := func(x float64, p []float64) float64 {
        res := 0.0
        for i := 0; i < len(p); i++ {
            res += math.Pow(x, float64(i)) * p[i]
        }
        return res
    }
```

```
    delta := 0.0
    for i := 0; i < len(x); i++ {
        delta += math.Pow(POLY(x[i], a)-y[i], 2.0)
    }
    return delta
}
```

derivative.go:

```
func DX(table [][]float64, x float64, i int) float64 {
    return (table[1][i+1]-table[1][i])/(table[0][i+1]-table[0][i]) +
((table[1][i+2]-table[1][i+1])/(table[0][i+2]-table[0][i+1])-(table[1]
[i+1]-table[1][i])/(table[0][i+1]-table[0][i]))/(table[0][i+2]-table[0]
[i])*(2*x-table[0][i]-table[0][i+1])
}

func D2X(table [][]float64, x float64, i int) float64 {
    return 2 * ((table[1][i+2]-table[1][i+1])/(table[0][i+2]-table[0]
[i+1]) - (table[1][i+1]-table[1][i])/(table[0][i+1]-table[0][i])) /
(table[0][i+2] - table[0][i])
}
```

integral.go:

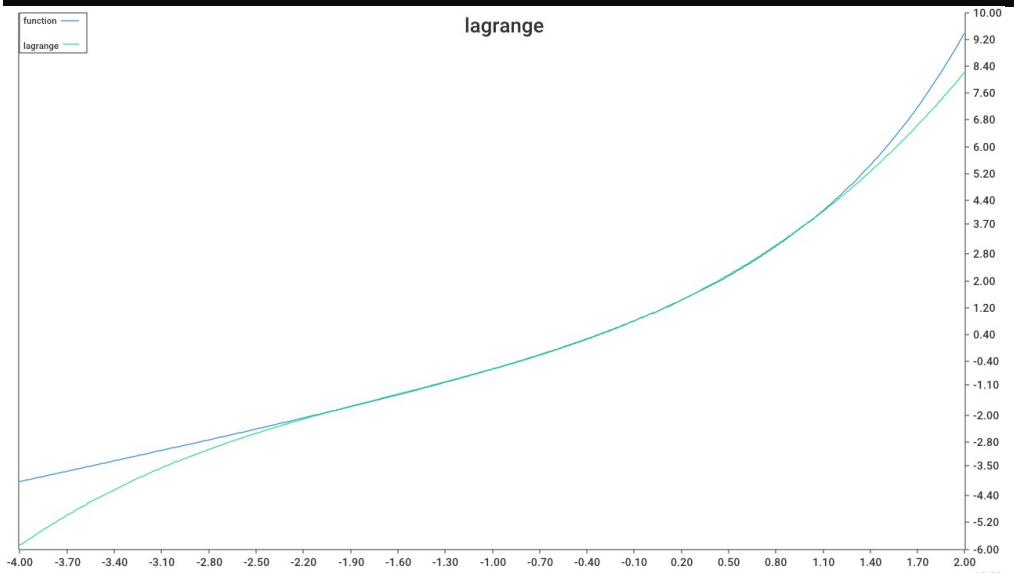
```
func IntegralRect(f func(float64) float64, a, b, h float64) float64 {
    x1 := a
    x2 := a + h
    res := 0.0
    for x1 < b {
        res += f((x1 + x2) / 2)
        x1 = x2
        x2 += h
    }
    return res * h
}

func IntegralTrapezoid(f func(float64) float64, a, b, h float64) float64
{
    x1 := a
    x2 := a + h
    res := 0.0
    for x1 < b {
        res += (f(x1) + f(x2))
        x1 = x2
        x2 += h
    }
    return res * (h / 2)
}
```

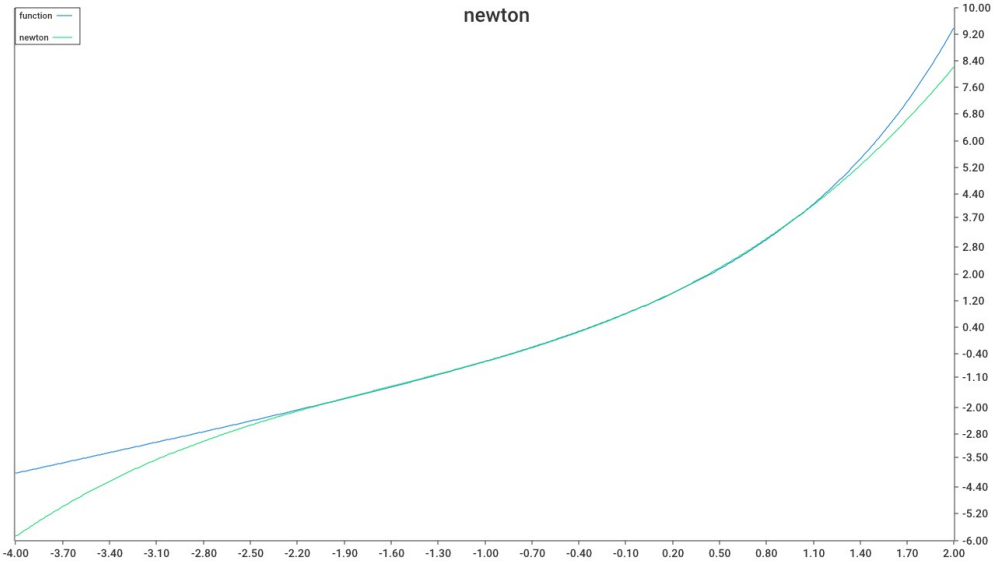
```go
func IntegralSimpson(f func(float64) float64, a, b, h float64) float64 {
    x := a + h
    res := 0.0
    for x < b {
        res += f(x-h) + 4*f(x) + f(x+h)
        x += h * 2
    }
    return res * (h / 3)
}

func RRRMethod(Fh, Fkh, k, p float64) float64 {
    return (Fh - Fkh) / (math.Pow(k, p) - 1.0)
}
```

Пример работы:

3.1)

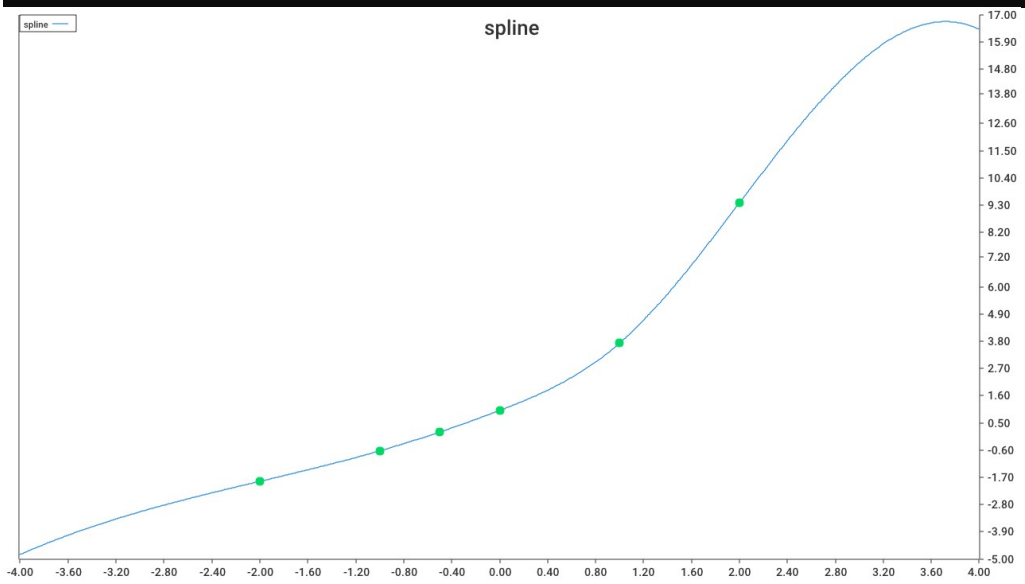
```
C:\Users\sinyu\Desktop\numeric\lab3>go run main.go < 1.txt
Select lab 3.X:
1: interpolation
2: spline
3: LSM
4: derivative
5: integral
3.1-----Interpolation-----
Size:
X:
X*:

Lagrange:
1 + 2.060770382187675*x^1 + 0.5430806348152437*x^2 + 0.11443081145612644*x^3
f(x*) =  0.0910811161779576
error =  0.015449543534675828

Newton:
1 + 2.060770382187675*x^1 + 0.5430806348152437*x^2 + 0.11443081145612655*x^3
f(x*) =  0.09108111617795758
error =  0.015449543534675841
```





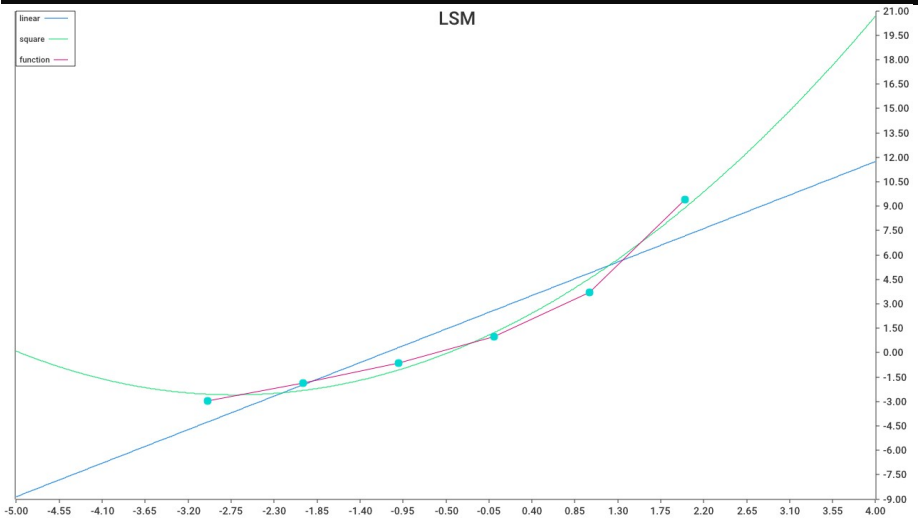
3.2)

```
C:\Users\sinyu\Desktop\numeric\lab3>go run main.go < 2.txt
Select lab 3.X:
1: interpolation
2: spline
3: LSM
4: derivative
5: integral
3.2-----Spline-----
Size:
X:
F:
X*:
|   a    |   b    |   c    |   d    |
|-1.86470|1.15042|0.00000|0.08216|
|-0.63212|1.39690|0.24648|-0.01125|
|1.00000|1.85609|0.21272|0.64949|
|3.71830|4.23000|2.16120|-0.72040|
f(x*) =  0.12654089285714284
```



3.3)

```
C:\Users\sinyu\Desktop\numeric\lab3>go run main.go < 3.txt
Select lab 3.X:
1: interpolation
2: spline
3: LSM
4: derivative
5: integral
3.3-----LSM-----
Size:
X:
Y:
p = 1:
2.587362666666667 + 2.287932*x^1
Error:  11.454877259413331

p = 2:
1.2126302857142865 + 2.8034566428571424*x^1 + 0.5155246428571426*x^2
Error:  1.5329593834085702
```



3.4)

```
C:\Users\sinyu\Desktop\numeric\lab3>go run main.go < 4.txt
Select lab 3.X:
1: interpolation
2: spline
3: LSM
4: derivative
5: integral
3.4-----Derivative-----
Size:
X:
Y:
X*:
D1:  2.0288
D2:  0.22000000000000242
```

3.5)

```
C:\Users\sinyu\Desktop\numeric\lab3>go run main.go
Select lab 3.X:
1: interpolation
2: spline
3: LSM
4: derivative
5: integral
5
3.5-----Integral-----
Rect method (h1):  0.015784519894108937
Rect method (h2):  0.015816058350960477

Trapezoid method (h1):  0.015916053921568626
Trapezoid method (h2):  0.01585028690783878

Simpson method (h1):  0.015839460784313725
Simpson method (h2):  0.01582836456992883

Rect err:  4.2051275802053777e-05
Trapezoid err:  -8.768935163979326e-05
Simpson err:  -1.479495251319185e-05
```

Лабораторная работа №4

Задание:

4.1. Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки . С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

4.2. Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

Код:

Программа реализована на языке программирования Go.

Cauchy.go:

```
func DiffEuler(system []func(x, y, z float64) float64, yKey, zKey
[]float64, xl, xr, h float64) ([]float64, []float64, []float64) {
    var x, y, z []float64
    x = append(x, xl)
    y = append(y, yKey[1])
    z = append(z, zKey[1])
    count := 0
    for x[count]+0.0000001 < xr {
        x = append(x, x[count]+h)
        y = append(y, y[count]+h*system[0](x[count], y[count], z[count]))
        z = append(z, z[count]+h*system[1](x[count], y[count], z[count]))
        count++
    }
    return x, y, z
}

func DiffRungeKutt(system []func(x, y, z float64) float64, yKey, zKey
[]float64, xl, xr, h float64) ([]float64, []float64, []float64) {
    var x, y, z []float64
    x = append(x, xl)
    y = append(y, yKey[1])
    z = append(z, zKey[1])
    count := 0
    for x[count]+0.0000001 < xr {
        K1 := h * system[0](x[count], y[count], z[count])
        L1 := h * system[1](x[count], y[count], z[count])
        K2 := h * system[0](x[count]+h/2, y[count]+K1/2, z[count]+L1/2)
        L2 := h * system[1](x[count]+h/2, y[count]+K1/2, z[count]+L1/2)
        K3 := h * system[0](x[count]+h/2, y[count]+K2/2, z[count]+L2/2)
        L3 := h * system[1](x[count]+h/2, y[count]+K2/2, z[count]+L2/2)
        K4 := h * system[0](x[count]+h, y[count]+K3, z[count]+L3)
        L4 := h * system[1](x[count]+h, y[count]+K3, z[count]+L3)
        deltaY := (K1 + 2*K2 + 2*K3 + K4) / 6
        deltaZ := (L1 + 2*L2 + 2*L3 + L4) / 6
        x = append(x, x[count]+h)
        y = append(y, y[count]+deltaY)
        z = append(z, z[count]+deltaZ)
        count++
    }
    return x, y, z
}

func DiffAdams(system []func(x, y, z float64) float64, yKey, zKey
```

```
[]float64, xl, xr, h float64) ([]float64, []float64, []float64) {
    x, y, z := DiffRungeKutt(system, yKey, zKey, xl, xl+4*h, h)
    count := len(x) - 1
    for x[count]+0.0000001 < xr {
        x = append(x, x[count]+h)
        y = append(y, y[count]+(h/24)*(55*system[0](x[count], y[count],
z[count])-59*system[0](x[count-1], y[count-1], z[count-1])+37*system[0]
(x[count-2], y[count-2], z[count-2])-9*system[0](x[count-3], y[count-3],
z[count-3])))
        z = append(z, z[count]+(h/24)*(55*system[1](x[count], y[count],
z[count])-59*system[1](x[count-1], y[count-1], z[count-1])+37*system[1]
(x[count-2], y[count-2], z[count-2])-9*system[1](x[count-3], y[count-3],
z[count-3])))
        count++
    }
    return x, y, z
}

func RRRmethod(y1, y2 []float64, m float64) float64 {
    res := 0.0
    for i := 0; i < len(y1); i++ {
        res = math.Max(res, math.Abs(y1[i]-y2[i*2])/math.Pow(2.0, m))
    }
    return res
}

func AbsoluteError(y1, y2 []float64) float64 {
    res := 0.0
    for i := 0; i < len(y1); i++ {
        res = math.Max(res, math.Abs(y1[i]-y2[i]))
    }
    return res
}
```

boundary.go:

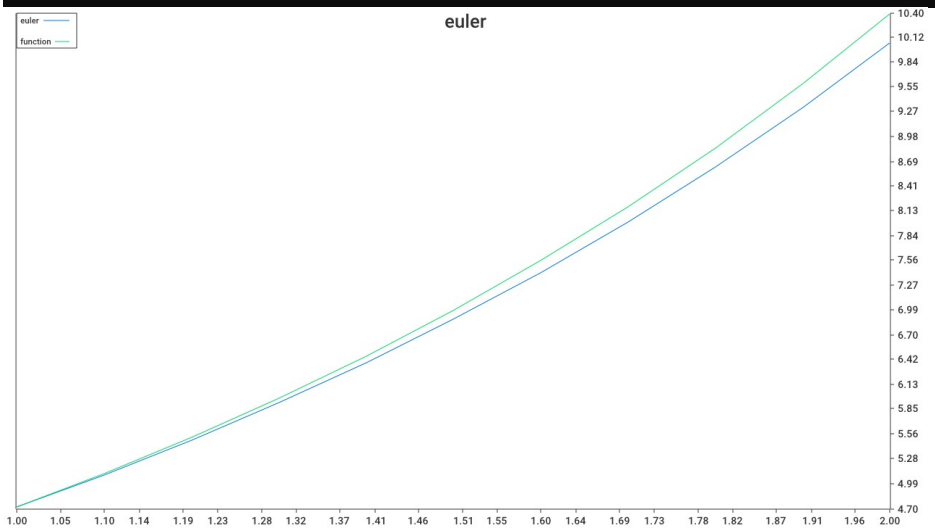
```
func DiffShooting(system []func(x, y, z float64) float64, xl, xr, al, bl,
yl, ar, br, yr, h, eps float64) ([]float64, []float64) {
    n0, n1, n2 := 1.0, 0.8, 2.0
    var x1, y1, z1 []float64
    for math.Abs(n2) > eps {
        _, y0, z0 := cauchy.DiffAdams(
            system, append(make([]float64, 0, 2), xl, n0),
            append(make([]float64, 0, 2), xl, (yl-(al*n0))/bl), xl, xr,
h)
        x1, y1, z1 = cauchy.DiffAdams(
            system, append(make([]float64, 0, 2), xl, n1),
            append(make([]float64, 0, 2), xl, (yl-(al*n1))/bl), xl, xr,
h)
```

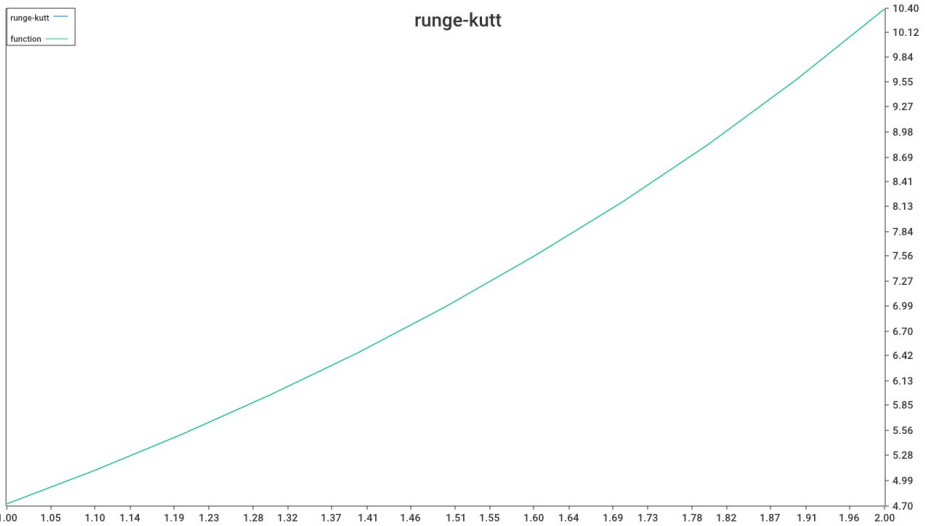
```
        f0 := ar*y0[len(y0)-1] + br*z0[len(z0)-1] - yr
        f1 := ar*y1[len(y1)-1] + br*z1[len(z1)-1] - yr
        n2 = n1 - (n1-n0)/(f1-f0)*f1
        n0 = n1
        n1 = n2
    }
    return x1, y1
}

func DiffFinite(system []func(x, y, z float64) float64, xl, xr, al, bl,
yl, ar, br, yr, h float64) ([]float64, []float64) {
    p := func(x float64) float64 {
        return -1 * system[1](x, 0, 1)
    }
    q := func(x float64) float64 {
        return -1 * system[1](x, 1, 0)
    }
    f := func(x float64) float64 {
        return system[0](x, 0, 0)
    }

    var a, b, c, d, x []float64
    x = append(x, xl)
    count := 0

    a = append(a, 0)
    b = append(b, al-bl/h)
    c = append(c, bl/h)
    d = append(d, yl)

    for x[count]+0.0000001 < xr-h {
        x = append(x, x[count]+h)
        count++
        a = append(a, 1-p(x[count])*h*0.5)
        b = append(b, -2+h*h*q(x[count]))
        c = append(c, 1+p(x[count])*h*0.5)
        d = append(d, h*h*f(x[count]))
    }

    x = append(x, xr)
    a = append(a, -br/h)
    b = append(b, ar+br/h)
    c = append(c, 0)
    d = append(d, yr)

    y := tridiagonal.SolveTridiagonal(a, b, c, d, len(b))
    return x, y
}
```

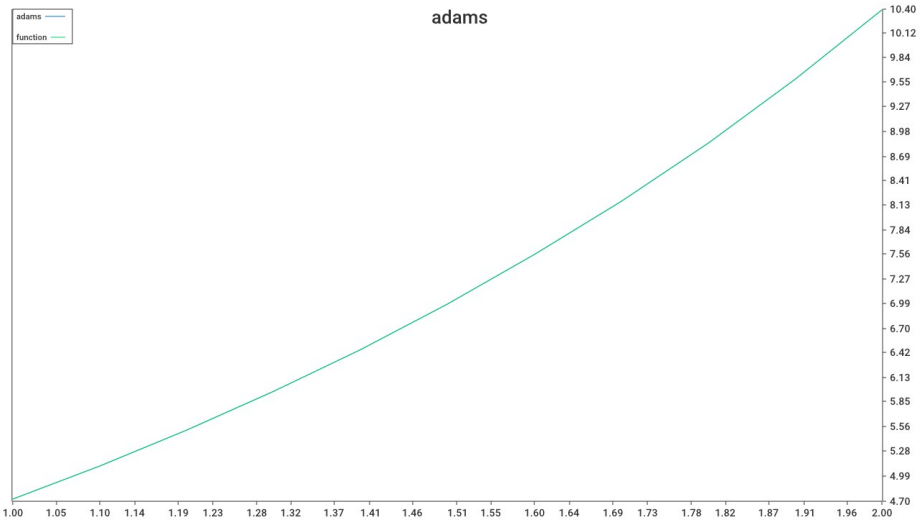
Пример работы:

4.1)

```
C:\Users\sinyu\Desktop\numeric\lab4>go run main.go
Select lab 4.X:
1: Cauchy
2: boundary
1
Euler error (RRR):  0.08094397019683619
Runge-Kutt error (RRR):  3.31039355150331e-07
Adams error(RRR):  7.65683920034288e-06

Euler error:  0.33853310193816455
Runge-Kutt error:  5.665779729824294e-06
Adams error:  0.0001345000347363623
```

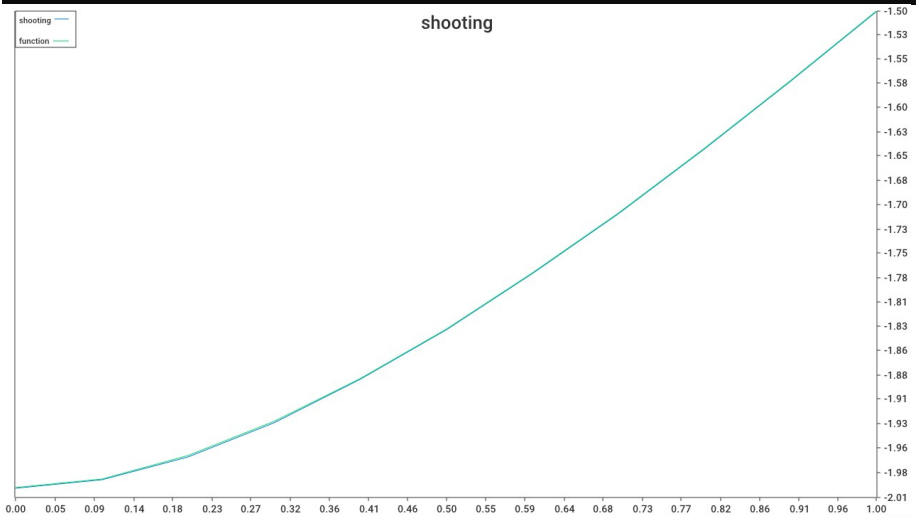


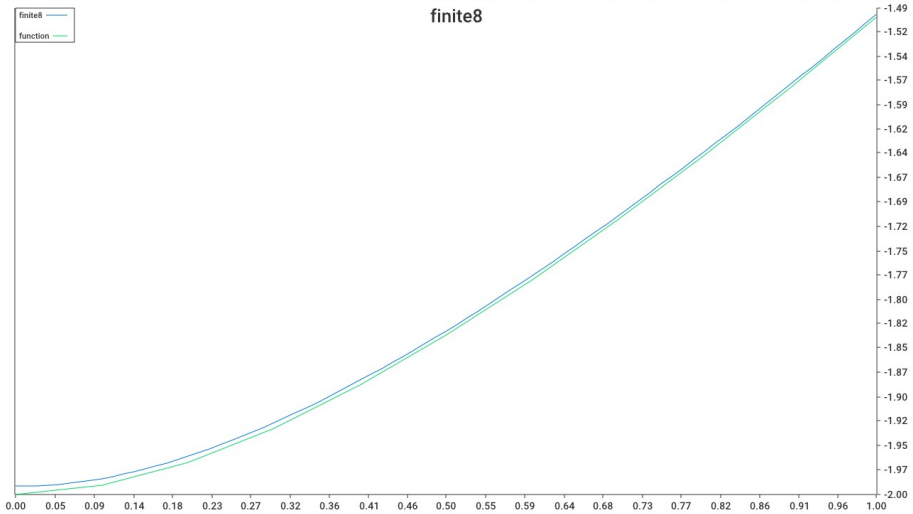




4.2)

```
C:\Users\sinyu\Desktop\numeric\lab4>go run main.go
Select lab 4.X:
1: Cauchy
2: boundary
2
shooting error (RRR):  0.0002522184476245748
finite error (RRR):  0.008822416696739777

shooting error:  0.0010050311729052908
finite error:  0.06934896251992084
```





Выводы:

При выполнении лабораторных работ были изучены многие численные методы, позволяющие решить различный спектр задач. Также во время выполнения был освоен в бОльшей степени язык программирования Go, были написаны многие вспомогательные функции (такие как перемножение матриц, перемножение многочленов), а также освоен пакет для построения графиков на языке Go.