

package main

import (

"fmt"

"math"

"sort"

)

type Interval struct {

l, r float64

}

func createVector(numCols, V int, rad float64) []Interval {

vector := make([]Interval, numCols)

for i := 1; i < numCols; i++ {

point := (2.7 \* float64(V)) / math.Log(float64(6+i))

vector[i] = Interval{l: point - rad, r: point + rad}

}

print(vector)

return vector

}

func createArr(numRows, numCols, V int, rad float64) [][]Interval {

intervals := make([][]Interval, numRows)

for i := 1; i < numRows; i++ {

intervals[i] = make([]Interval, numCols)

}

for i := 1; i < numRows; i++ {

for j := i; j < numCols; j++ {

if i == j {

intervals[i][j].l = (31.0 + float64(V)\*math.Sin(float64(i))) - rad

intervals[i][j].r = (31.0 + float64(V)\*math.Sin(float64(i))) + rad

} else {

intervals[i][j].l = (0.01\*float64(V) + math.Log(float64(i+j))) - rad

intervals[i][j].r = (0.01\*float64(V) + math.Log(float64(i+j))) + rad

intervals[j][i].l = intervals[i][j].l

intervals[j][i].r = intervals[i][j].r

}

}

}

return intervals

}

func intervalMultiplication(X, Y Interval) Interval {

tmp := make([]float64, 4)

tmp[0] = X.l \* Y.l

tmp[1] = X.l \* Y.r

tmp[2] = X.r \* Y.l

tmp[3] = X.r \* Y.r

sort.Float64s(tmp)

res := Interval{l: tmp[0], r: tmp[len(tmp)-1]}

return res

}

func intervalAddition(X, Y Interval) Interval {

return Interval{l: X.l + Y.l, r: X.r + Y.r}

}

func multiplication(A [][]Interval, B []Interval, rad float64) []Interval {

numRows := len(A)

numCols := len(A[1])

if numCols != len(B) {

panic("Количество столбцов матрицы должно быть равно длине вектора")

}

var tmp Interval

result := make([]Interval, numRows)

for i := 1; i < numRows; i++ {

for j := 1; j < numCols; j++ {

tmp = intervalAddition(tmp, intervalMultiplication(A[i][j], B[j]))

}

result[i] = tmp

tmp = Interval{l: 0, r: 0}

}

return result

}

func main() {

rad := 0.005

V := 3

numRows, numCols := 4, 4

n, m := numRows+1, numCols+1

A := createArr(n, m, V, rad)

B := createVector(m, V, rad)

C := multiplication(A, B, rad)

fmt.Println()

fmt.Print("Матрица A\n")

for i := 1; i < n; i++ {

for j := 1; j < m; j++ {

fmt.Printf("[%0.3f, %0.3f] ", A[i][j].l, A[i][j].r)

}

fmt.Println()

}

fmt.Println()

fmt.Print("Матрица B\n")

for i := 1; i < n; i++ {

fmt.Printf("[%0.3f, %0.3f]\n", B[i].l, B[i].r)

}

fmt.Println()

fmt.Print("Матрица C в виде интервалов\n")

for i := 1; i < n; i++ {

fmt.Printf("[%0.3f, %0.3f]\n", C[i].l, C[i].r)

}

fmt.Println()

fmt.Print("Матрица C в виде середина-радиус\n")

fmt.Print("Середина, Радиус\n")

for i := 1; i < n; i++ {

var tmp float64 = C[i].l + (C[i].r-C[i].l)/2

fmt.Printf("[%0.3f, %0.3f]\n", tmp, tmp-C[i].l)

}

}

