

Код программы  
package main

import (

"fmt"

"math"

"sort"

)

type Interval struct {

L, R float64

}

func NegativeInterval(X Interval) Interval {

res := Division(X, Interval{L: -1, R: -1})

return res

}

// Возвращает большее значение

func Comparison(X, Y Interval) Interval {

var res Interval

if X.R < Y.L {

return Y

} else if X.L > Y.R {

return X

}

return res

}

func Multiplication(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 Division(X, Y Interval) Interval {

if Comparison(Multiplication(X, Y), Interval{L: 0, R: 0}) == Multiplication(X, Y) {

tmp := make([]float64, 2)

tmp[0] = X.L / Y.L

tmp[1] = X.R / Y.R

sort.Float64s(tmp)

res := Interval{L: tmp[0], R: tmp[1]}

return res

} else if Comparison(Multiplication(X, Y), Interval{L: 0, R: 0}) != Multiplication(X, Y) {

tmp := make([]float64, 2)

tmp[0] = X.L / Y.R

tmp[1] = X.R / Y.L

sort.Float64s(tmp)

res := Interval{L: tmp[0], R: tmp[1]}

return res

} else if X.L <= 0 && X.R >= 0 && Y.L > 0 && Y.R > 0 {

tmp := 1 / Y.L

return Interval{L: tmp \* X.L, R: tmp \* X.R}

} else if X.L <= 0 && X.R >= 0 && Y.L < 0 && Y.R < 0 {

tmp := 1 / Y.R

return Interval{L: tmp \* X.L, R: tmp \* X.R}

} else {

panic("Данные вычисления не предусмотренны библиотекой")

}

}

func Addition(X, Y Interval) Interval {

tmp := make([]float64, 2)

tmp[0] = X.L + Y.L

tmp[1] = X.R + Y.R

sort.Float64s(tmp)

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

return res

}

func Substraction(X, Y Interval) Interval {

tmp := make([]float64, 2)

tmp[0] = X.L - Y.L

tmp[1] = X.R - Y.R

sort.Float64s(tmp)

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

return res

}

func CreateMatrix(rad float64, V int, M int) ([]Interval, []Interval, []Interval) {

C := make([]Interval, M)

B := make([]Interval, M)

A := make([]Interval, M)

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

var tmp float64 = 10\*float64(V) + float64(i)/float64(V)

B[i] = Interval{L: tmp - rad, R: tmp + rad}

if i < M-1 {

var tmp float64 = 0.4 \* math.Cos(float64(i)) / float64(V)

C[i] = Interval{L: tmp - rad, R: tmp + rad}

} else {

C[i] = Interval{L: 0, R: 0}

}

if i > 1 {

var tmp float64 = 0.3 \* math.Sin(float64(i)) / float64(V)

A[i] = Interval{L: tmp - rad, R: tmp + rad}

} else {

A[i] = Interval{L: 0, R: 0}

}

}

return C, B, A

}

func CreateVector(rad float64, V int, M int) []Interval {

D := make([]Interval, M)

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

tmp := 1.3 + float64(i)/float64(V)

D[i] = Interval{L: tmp - rad, R: tmp + rad}

}

return D

}

func SolveTridiagonalSystem(C, B, A, D []Interval) []Interval {

n := len(B)

alpha := make([]Interval, n)

beta := make([]Interval, n)

x := make([]Interval, n)

alpha[1] = Division(NegativeInterval(C[1]), B[1])

beta[1] = Division(D[1], B[1])

// Прямой Ход

for i := 2; i < n-1; i++ {

alpha[i] = Division(NegativeInterval(C[i]), Addition(B[i], Multiplication(A[i], alpha[i-1])))

beta[i] = Division(Substraction(D[i], Multiplication(A[i], beta[i-1])), Addition(B[i], Multiplication(A[i], alpha[i-1])))

}

beta[n-1] = Division(Substraction(D[n-1], Multiplication(A[n-1], beta[n-2])), Addition(B[n-1], Multiplication(A[n-1], alpha[n-2])))

// Обратный ход

x[n-1] = beta[n-1]

for i := n - 2; i > 0; i-- {

x[i] = Addition(Multiplication(alpha[i], x[i+1]), beta[i])

}

return x

}

func CheckAnswer(C, B, A, D, X []Interval) ([]Interval, Interval) {

n := len(B)

res := make([]Interval, n)

// Умножение матрицы на вектор решения

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

res[i] = Multiplication(B[i], X[i])

if i < n-1 {

res[i] = Addition(res[i], Multiplication(C[i], X[i+1]))

}

if i > 1 {

res[i] = Addition(res[i], Multiplication(A[i], X[i-1]))

}

}

NevVector := make([]Interval, n)

// Вычисление вектора невязки

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

NevVector[i] = Substraction(D[i], res[i])

}

var norm Interval

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

norm = Addition(norm, Interval{L: math.Pow(NevVector[i].L, 2), R: math.Pow(NevVector[i].R, 2)})

}

return NevVector, norm

}

func PrintVectorInRange(arr []Interval, r, count int, name string) {

fmt.Println("\n\t\t", name)

for i := r; i < r+count; i++ {

fmt.Printf("[%15.6e, %15.6e] ", arr[i].L, arr[i].R)

fmt.Println()

}

}

func PrintVector(arr []Interval, name string) {

n := len(arr)

fmt.Println("\n\t\t", name)

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

fmt.Printf("[%15.6e, %15.6e] ", arr[i].L, arr[i].R)

fmt.Println()

}

}

func main() {

fmt.Println()

rad := 0.01

V := 3

M := 8

C, B, A := CreateMatrix(rad, V, M+1)

D := CreateVector(rad, V, M+1)

X := SolveTridiagonalSystem(C, B, A, D)

res, norm := CheckAnswer(C, B, A, D, X)

fmt.Println("M = 8")

PrintVector(C, "Вектор C")

PrintVector(B, "Вектор B")

PrintVector(A, "Вектор A")

PrintVector(D, "Вектор D")

PrintVector(X, "Вектор X")

PrintVector(res, "Вектор невязки")

fmt.Print("\n\tНорма вектор невязки\n")

fmt.Printf("[%15.6e, %15.6e]\n", norm.L, norm.R)

M = 3000000

r := 500001

count := 4

C, B, A = CreateMatrix(rad, V, M+1)

D = CreateVector(rad, V, M+1)

X = SolveTridiagonalSystem(C, B, A, D)

res, norm = CheckAnswer(C, B, A, D, X)

fmt.Print("\n\nM = 3000000\n")

PrintVectorInRange(C, r, count, "Вектор C")

PrintVectorInRange(B, r, count, "Вектор B")

PrintVectorInRange(A, r, count, "Вектор A")

PrintVectorInRange(D, r, count, "Вектор D")

PrintVectorInRange(X, r, count, "Вектор X")

PrintVectorInRange(res, r, count, "Вектор невязки")

fmt.Print("\n\tНорма вектор невязки\n")

fmt.Printf("[%15.6e, %15.6e]\n ", norm.L, norm.R)

}

