```go
package main

import (
  "fmt"
  "math"
)

func inicializarVetorSolucaoZero(c configuration) (vetSol [][]float64) {
  for i := 0; i < c.systemOrder; i++ {
    line := []float64{1}
    vetSol = append(vetSol, line)
  }
  return vetSol
}

func achaProxVetSolDadoVetSolAtualEConfiguracoes(vetSolVelho [][]float64, c
configuration) (vetSolNovo [][]float64) {
  for i := 0; i < c.systemOrder; i++ {
    var xi []float64
    var soma float64 = 0
    for j := 0; j < c.systemOrder; j++ {
      if i == j {
        continue
      }
      soma += c.matrixA[i][j] * vetSolVelho[j][0]
    }
    xiNum := (c.vectorB[i][0] - soma) / c.matrixA[i][i]
    xi = append(xi, xiNum)
    vetSolNovo = append(vetSolNovo, xi)
  }
  return vetSolNovo
}

func MediaEuclidiana(vet [][]float64) (res float64) {
  var soma float64 = 0
  for i := 0; i < len(vet); i++ {
    soma += math.Pow(vet[i][0], 2)
```

```
37    }
38    res = math.Sqrt(soma)
39    return res
40  }
41
42  //subtraiVetores ... retorna o resultado de a-b
43  func subtraiVetores(a, b [][]float64) (res [][]float64) {
44    for i := 0; i < len(a); i++ {
45      num := a[i][0] - b[i][0]
46      res = append(res, []float64{num})
47    }
48    return res
49  }
50
51  func CalcResiduo(vetSolNovo, vetSolVelho [][]float64) (residuo float64) {
52    resSub := subtraiVetores(vetSolNovo, vetSolVelho)
53    dividendo := MediaEuclidiana(resSub)
54    divisor := MediaEuclidiana(vetSolNovo)
55    residuo = dividendo / divisor
56    return residuo
57  }
58
59  func chechaSeMatrizEDiagonalPrincipal(c configuration) (podeAplicar bool) {
60    for i := 0; i < c.systemOrder; i++ {
61      aii := c.matrixA[i][i]
62      var soma float64 = 0
63      for j := 0; j < c.systemOrder; j++ {
64        if i == j {
65          continue
66        }
67        soma += c.matrixA[i][j]
68      }
69      if aii < soma {
70        return false
71      }
72    }
73    return true
```

```
 74 }
 75
 76 func SolucaoPeloProcedimentoIterativoDeJacobi(c configuration) (vetSol
    [][]float64) {
 77   fmt.Println("Iniciando solução pelo Procedimento Iterativo de Jacobi")
 78   podeAplicar := chechaSeMatrizEDiagonalPrincipal(c)
 79   if !podeAplicar {
 80     panic("O método Iterativo de Jacobi não pode ser aplicado a matriza dada
    pois esta não é diagonal dominante.")
 81   }
 82   vetSolAnterior := inicializarVetorSolucaoZero(c)
 83   vetSol = achaProxVetSolDadoVetSolAtualEConfiguracoes(vetSolAnterior, c)
 84   residuo := CalcResiduo(vetSol, vetSolAnterior)
 85
 86   //Printando output
 87   vetSolAnteriorString := CreateMatrixString(vetSolAnterior)
 88   stringDepuracao := fmt.Sprintf("VetorSolucao:\n%s\n", vetSolAnteriorString)
 89   stringDepuracao2 := fmt.Sprintf("Residuo:\n%v\n", residuo)
 90   Pw(OUTPUT_FILE_PATH, stringDepuracao)
 91   Pw(OUTPUT_FILE_PATH, stringDepuracao2)
 92   Pw(OUTPUT_FILE_PATH, SEPARADOR)
 93
 94   for residuo > c.TOLm {
 95     vetSolAnterior = vetSol
 96     vetSol = achaProxVetSolDadoVetSolAtualEConfiguracoes(vetSolAnterior, c)
 97     residuo = CalcResiduo(vetSol, vetSolAnterior)
 98
 99     //Printando output loop
100     stringDepuracao = fmt.Sprintf("VetorSolucao:\n%s\n", vetSolAnteriorString)
101     stringDepuracao2 = fmt.Sprintf("Residuo:\n%v\n", residuo)
102     Pw(OUTPUT_FILE_PATH, stringDepuracao)
103     Pw(OUTPUT_FILE_PATH, stringDepuracao2)
104     Pw(OUTPUT_FILE_PATH, SEPARADOR)
105   }
106
107   //Printando output final
108   vetSolString := CreateMatrixString(vetSol)
```

```
109   stringDepuracao = fmt.Sprintf("VetorSolucaoFinal:\n%s\n", vetSolString)
110   stringDepuracao2 = fmt.Sprintf("Residuo Final:\n%v\n", residuo)
111   Pw(OUTPUT_FILE_PATH, stringDepuracao)
112   Pw(OUTPUT_FILE_PATH, stringDepuracao2)
113   Pw(OUTPUT_FILE_PATH, SEPARADOR)
114   return vetSol
115 }
116
```