```go
package main

import (
  "fmt"
  "math"
)

func checaSePodeAplicarOMetodoDeJacobi(c configuration) (podeAplicar bool) {
  matrizSimetrica := checaSeMatrizESimetrica(c)
  return matrizSimetrica
}

func criaMatrizIdentidade(c configuration) (I [][]float64) {
  numOfRows := len(c.matrixA)
  numOfCols := len(c.matrixA[0])
  I = InitializeMatrixWithZeros(numOfRows, numOfCols)
  for i := 0; i < numOfRows; i++ {
    for j := 0; j < numOfCols; j++ {
      if i == j {
        I[i][j] = 1
      }
    }
  }
  return I
}

func achaMaiorElementoForaDaDiagPrincipal(matrix [][]float64) (elem float64,
ir, jr int) {
  numOfRows := len(matrix)
  numOfCols := len(matrix[0])
  elem = 0
  for i := 0; i < numOfRows; i++ {
    for j := 0; j < numOfCols; j++ {
      if i == j {
        continue
      }
      abs := math.Sqrt(math.Pow(matrix[i][j], 2))
```

```
37        if abs > elem {
38          elem = abs
39          ir = i
40          jr = j
41        }
42      }
43    }
44    return elem, ir, jr
45  }
46
47  func CalcularMatrizPk(matrix [][]float64, ir, jr int) (matrizPk [][]float64) {
48    var teta float64
49    if matrix[ir][ir] != matrix[jr][jr] {
50      teta = 0.5 * math.Atan(2*matrix[ir][jr]/(matrix[ir][ir]-matrix[jr][jr]))
51    } else {
52      teta = math.Pi / 4
53    }
54    c := configuration{
55      systemOrder: 0,
56      ICOD:        0,
57      IDET:        0,
58      matrixA:     matrix,
59      vectorB:     [][]float64{},
60      TOLm:        0,
61    }
62    I := criaMatrizIdentidade(c)
63    I[ir][ir] = math.Cos(teta)
64    I[jr][jr] = math.Cos(teta)
65    if ir > jr {
66      I[ir][jr] = math.Sin(teta)
67      I[jr][ir] = -1 * math.Sin(teta)
68    } else {
69      I[ir][jr] = -1 * math.Sin(teta)
70      I[jr][ir] = math.Sin(teta)
71    }
72    return I
73  }
```

```
 74
 75 func achaMatrizTransposta(m [][]float64) (mT [][]float64) {
 76   numOfRows := len(m)
 77   numOfCols := len(m[0])
 78   mT = InitializeMatrixWithZeros(numOfRows, numOfCols)
 79   for i := 0; i < numOfRows; i++ {
 80     for j := 0; j < numOfCols; j++ {
 81       mT[i][j] = m[j][i]
 82     }
 83   }
 84   return mT
 85 }
 86
 87 func AchaAutovaloresEAutovetoresViaMetodoDeJacobi(c configuration)
    (autovalores, autovetores [][]float64) {
 88   fmt.Println("Starting solution via MetodoDeJacobi")
 89   podeAplicar := checaSePodeAplicarOMetodoDeJacobi(c)
 90   if !podeAplicar {
 91     fmt.Println("A matriz A não é simétrica, o método de Jacobi não pode ser
    aplicado")
 92     panic("Método escolhido não serve para matriz de input.")
 93   }
 94   //Passo1
 95   A1 := c.matrixA
 96   X1 := criaMatrizIdentidade(c)
 97
 98   //passo2
 99   iteracao := 0
100
101   //Passo2.1
102   maiorElemento, im, jm := achaMaiorElementoForaDaDiagPrincipal(A1)
103
104   //Passo2.2
105   pk := CalcularMatrizPk(A1, im, jm)
106   pkT := achaMatrizTransposta(pk)
107   step0, _ := MultiplyMatrices(pkT, A1)
108   Anovo, _ := MultiplyMatrices(step0, pk)
```

```
109   Xnovo, _ := MultiplyMatrices(X1, pk)
110
111   //Imprime valores da iteração
112   Pw(OUTPUT_FILE_PATH, fmt.Sprintf("Iteracao %v\n", iteracao))
113   Pw(OUTPUT_FILE_PATH, fmt.Sprintf("A1:\n%s\nX1:\n%s\n",
    CreateMatrixString(A1), CreateMatrixString(X1)))
114   Pw(OUTPUT_FILE_PATH, fmt.Sprintf("Maior elemento: A1(%v%v) %v\n", im, jm,
    maiorElemento))
115   Pw(OUTPUT_FILE_PATH, fmt.Sprintf("Matriz pk:\n%s\n",
    CreateMatrixString(pk)))
116   Pw(OUTPUT_FILE_PATH, fmt.Sprintln(SEPARADOR))
117
118   //Atualiza valores
119   A1 = Anovo
120   X1 = Xnovo
121
122   //Passo3
123   for maiorElemento > c.TOLm {
124     iteracao++
125
126     maiorElemento, im, jm = achaMaiorElementoForaDaDiagPrincipal(A1)
127     pk := CalcularMatrizPk(A1, im, jm)
128     pkT := achaMatrizTransposta(pk)
129     step0, _ := MultiplyMatrices(pkT, A1)
130     Anovo, _ := MultiplyMatrices(step0, pk)
131     Xnovo, _ := MultiplyMatrices(X1, pk)
132
133     Pw(OUTPUT_FILE_PATH, fmt.Sprintf("Iteracao %v\n", iteracao))
134     Pw(OUTPUT_FILE_PATH, fmt.Sprintf("A1:\n%s\nX1:\n%s\n",
    CreateMatrixString(A1), CreateMatrixString(X1)))
135     Pw(OUTPUT_FILE_PATH, fmt.Sprintf("Maior elemento: A1(%v%v) %v\n", im, jm,
    maiorElemento))
136     Pw(OUTPUT_FILE_PATH, fmt.Sprintf("Matriz pk:\n%s\n",
    CreateMatrixString(pk)))
137     Pw(OUTPUT_FILE_PATH, fmt.Sprintln(SEPARADOR))
138
139     A1 = Anovo
```

```
140     X1 = Xnovo
141   }
142
143   Pw(OUTPUT_FILE_PATH, fmt.Sprintf("----Resultado----\n"))
144   Pw(OUTPUT_FILE_PATH, fmt.Sprintf("A1:\n%s\nX1:\n%s\n",
    CreateMatrixString(A1), CreateMatrixString(X1)))
145   Pw(OUTPUT_FILE_PATH, fmt.Sprintf("Maior elemento: A1(%v%v) %v\n", im, jm,
    maiorElemento))
146   Pw(OUTPUT_FILE_PATH, fmt.Sprintf("Matriz pk:\n%s\n",
    CreateMatrixString(pk)))
147   Pw(OUTPUT_FILE_PATH, fmt.Sprintln(SEPARADOR))
148   return A1, X1
149 }
150
151 func achaInversaDeMatrizDiagonal(matrix [][]float64) (matrixInv [][]float64) {
152   numOfRows := len(matrix)
153   numOfColumns := len(matrix[0])
154   matrixInv = InitializeMatrixWithZeros(numOfRows, numOfColumns)
155   for i := 0; i < numOfRows; i++ {
156     for j := 0; j < numOfColumns; j++ {
157       if i == j {
158         matrixInv[i][j] = 1 / matrix[i][j]
159       }
160     }
161   }
162   return matrixInv
163 }
164
165 func SolucaoViaMetodoDeJacobi(c configuration) (sol [][]float64) {
166   lambda, teta := AchaAutovaloresEAutovetoresViaMetodoDeJacobi(c)
167   Pw(OUTPUT_FILE_PATH, fmt.Sprintf("lambda:\n%s\nteta:\n%s\n",
    CreateMatrixString(lambda), CreateMatrixString(teta)))
168   tetaT := achaMatrizTransposta(teta)
169   lambdaInv := achaInversaDeMatrizDiagonal(lambda)
170
171   Pw(OUTPUT_FILE_PATH, fmt.Sprintf("lambdaInversa:\n%s\n",
    CreateMatrixString(lambdaInv)))
```

```
172   Ystep, _ := MultiplyMatrices(tetaT, c.vectorB)
173   Y, _ := MultiplyMatrices(lambdaInv, Ystep)
174   Pw(OUTPUT_FILE_PATH, fmt.Sprintf("Y:\n%s\n", CreateMatrixString(Y)))
175   X, _ := MultiplyMatrices(teta, Y)
176   Pw(OUTPUT_FILE_PATH, fmt.Sprintf("X:\n%s\n", CreateMatrixString(X)))
177   return X
178 }
179
```