```go
package main

import (
  "fmt"
)

func achaProxVetSolDadoVetSolAtualEConfiguracoesGAUSSSEIDEL(vetSolVelho
[][]float64, c configuration) (vetSolNovo [][]float64) {
  for i := 0; i < c.systemOrder; i++ {
    var xi []float64
    var somaNovo float64 = 0
    var somaVelho float64 = 0

    for j := 0; j < i-1; j++ {
      somaNovo += c.matrixA[i][j] * vetSolNovo[j][0]
    }

    for j := i + 1; j < c.systemOrder; j++ {
      somaVelho += c.matrixA[i][j] * vetSolVelho[j][0]
    }

    xiNum := (c.vectorB[i][0] - somaNovo - somaVelho) / c.matrixA[i][i]
    xi = append(xi, xiNum)
    vetSolNovo = append(vetSolNovo, xi)
  }
  return vetSolNovo
}

func checaSePodeAplicarMetodoIterativoDeGaussSeidel(c configuration)
(podeAplicar bool) {
  matrizDiagonalPrincipal := chechaSeMatrizEDiagonalPrincipal(c)
  matrizPositivaDefinida := checaSeMatrizAEPositivaDefinida(c)
  matrizSimetrica := checaSeMatrizESimetrica(c)
  return matrizDiagonalPrincipal || matrizPositivaDefinida && matrizSimetrica
}

func SolucaoPeloProcedimentoIterativoDeGaussSeidel(c configuration) (vetSol
```

```
[][]float64) {
  fmt.Println("Iniciando solução pelo Procedimento Iterativo de Gauss Seidel")
  podeAplicar := checaSePodeAplicarMetodoIterativoDeGaussSeidel(c)
  if !podeAplicar {
    Pw(OUTPUT_FILE_PATH, "O ProcedimentoIterativoDeGaussSeidel não pode ser aplicado a matriza dada pois esta não é diagonal dominante nem positiva definida.")
    panic("O ProcedimentoIterativoDeGaussSeidel não pode ser aplicado a matriza dada pois esta não é diagonal dominante nem positiva definida.")
  }
  vetSolAnterior := inicializarVetorSolucaoZero(c)
  vetSol = achaProxVetSolDadoVetSolAtualEConfiguracoesGAUSSSEIDEL(vetSolAnterior, c)
  residuo := CalcResiduo(vetSol, vetSolAnterior)

  //Printando output
  vetSolAnteriorString := CreateMatrixString(vetSolAnterior)
  stringDepuracao := fmt.Sprintf("VetorSolucao:\n%s\n", vetSolAnteriorString)
  stringDepuracao2 := fmt.Sprintf("Residuo:\n%v\n", residuo)
  Pw(OUTPUT_FILE_PATH, stringDepuracao)
  Pw(OUTPUT_FILE_PATH, stringDepuracao2)
  Pw(OUTPUT_FILE_PATH, SEPARADOR)

  for residuo > c.TOLm {
    vetSolAnterior = vetSol
    vetSol = achaProxVetSolDadoVetSolAtualEConfiguracoes(vetSolAnterior, c)
    residuo = CalcResiduo(vetSol, vetSolAnterior)

    //Printando output loop
    stringDepuracao = fmt.Sprintf("VetorSolucao:\n%s\n", vetSolAnteriorString)
    stringDepuracao2 = fmt.Sprintf("Residuo:\n%v\n", residuo)
    Pw(OUTPUT_FILE_PATH, stringDepuracao)
    Pw(OUTPUT_FILE_PATH, stringDepuracao2)
    Pw(OUTPUT_FILE_PATH, SEPARADOR)
  }

  //Printando output final
```

```
68   vetSolString := CreateMatrixString(vetSol)
69   stringDepuracao = fmt.Sprintf("VetorSolucaoFinal:\n%s\n", vetSolString)
70   stringDepuracao2 = fmt.Sprintf("Residuo Final:\n%v\n", residuo)
71   Pw(OUTPUT_FILE_PATH, stringDepuracao)
72   Pw(OUTPUT_FILE_PATH, stringDepuracao2)
73   Pw(OUTPUT_FILE_PATH, SEPARADOR)
74   return vetSol
75 }
76
```