Apêndice G:

Código do aplicativo Simplex Formas v.1.1.0

Data de Publicação: 20 de março de 2025

```
#modificações sobre versão 25fev25
#correção lógica de formação IB impactando a aba Operações Iniciais

library(shiny)
library(mathjaxr)

# Função para criar o output da Forma Padrão
output_formapadrao <- function(input, num_variables, num_restrictions) {
  coefficients_z <- as.numeric(unlist(strsplit(input[["cost_vec"]], ",")))

  # Modify coefficients and objective function based on input$obj
  if (input$obj == "max") {
    coefficients_z <- -coefficients_z
    coefficients_z <- c(coefficients_z, rep(0, input$num_restrictions))
    cost_vec_z <- paste0("w = ", paste(paste0(coefficients_z, sprintf("x%d", 1:(num_variables + num_rest
  } else {
    coefficients_z <- c(coefficients_z, rep(0, input$num_restrictions))
    cost_vec_z <- paste0("z = ", paste(paste0(coefficients_z, sprintf("x%d", 1:(num_variables + num_rest
  }

  # Create an identity matrix for use in constructing coefficients_padrao
  matriz_identidade <- diag(input$num_restrictions)

  restrictions <- lapply(1:input$num_restrictions, function(i) {
    # Convert comma-separated values to numeric vector
    coefficients <- as.numeric(unlist(strsplit(input[[paste0("R", i)]], ",")))

    # Check if the direction is >=, and if true, negate the matriz_identidade[i,]
    if (input[[paste0("dir_", i)]] == ">=") {
      matriz_identidade[i,] <- -matriz_identidade[i,]
    }
    # Create coefficients_padrao with identity matrix
    coefficients_padrao <- c(coefficients[1:num_variables], matriz_identidade[i,], coefficients[(num_var

    # Check if the last element is less than 0, and if true, negate the entire vector
    if (coefficients_padrao[num_variables + num_restrictions + 1] < 0) {
      coefficients_padrao <- -coefficients_padrao
    }
    paste0("R", i, ": ", paste(paste0(coefficients_padrao[1:(num_variables + num_restrictions)], sprintf
           " = ", coefficients_padrao[num_variables + num_restrictions + 1])
  })

  HTML(paste0("<strong>", "min", "</strong> ", cost_vec_z, "<br>",
"<strong>sujeito a</strong><br>",
paste(restrictions, collapse = "<br>"), # Adicionar quebras de linha entre restrições
"<br>", paste(sprintf("x%d  0", 1:(num_variables + num_restrictions)), collapse = ", ")))
}

# Função para criar o output vetorcusto
output_vetorcusto <- function(input, num_variables, num_restrictions) {
  coefficients_z <- as.numeric(unlist(strsplit(input[["cost_vec"]], ",")))
  # Modify coefficients and objective function based on input$obj
  if (input$obj == "max") {
    coefficients_z <- -coefficients_z
```

```
    coefficients_z <- c(coefficients_z, rep(0, input$num_restrictions))
    cost_vec_z <- paste0("w = ", paste(paste0(coefficients_z, sprintf("x%d", 1:(num_variables + num_rest
  } else {
    coefficients_z <- c(coefficients_z, rep(0, input$num_restrictions))
    cost_vec_z <- paste0("z = ", paste(paste0(coefficients_z, sprintf("x%d", 1:(num_variables + num_rest
  }
  # Create a matrix with a single column
  #cost_vec_matrix <- matrix(coefficients_z, nrow =1 , ncol = length(coefficients_z), byrow = FALSE)

# Use HTML to format the matrix
  cost_vec_html <- paste0("<strong>vetor custo é: </strong>", "$$c^T=$$",
                          withMathJax("\\begin{bmatrix}",
                                      paste(coefficients_z, collapse = " & "),
                                      "\\end{bmatrix}"))

  # Return the HTML-formatted vector
  HTML(cost_vec_html)
}

# Função para criar o output vetorcusto não básico da iteração 1
output_vetorcusto_N <- function(input, num_variables, num_restrictions) {
  coefficients_z <- as.numeric(unlist(strsplit(input[["cost_vec"]], ",")))
  # Modify coefficients and objective function based on input$obj
  if (input$obj == "max") {
    coefficients_z <- -coefficients_z

    cost_vec_z <- paste0("w = ", paste(paste0(coefficients_z, sprintf("x%d", 1:(num_variables + num_rest
  } else {
    cost_vec_z <- paste0("z = ", paste(paste0(coefficients_z, sprintf("x%d", 1:(num_variables + num_rest
  }
  # Create a matrix with a single column
  #cost_vec_matrix <- matrix(coefficients_z, nrow =1 , ncol = length(coefficients_z), byrow = FALSE)

# Use HTML to format the matrix
  cost_vec_html <- paste0("<strong> </strong>", "$$c_N^T=$$",
                          withMathJax("\\begin{bmatrix}",
                                      paste(coefficients_z, collapse = " & "),
                                      "\\end{bmatrix}"))

  # Return the HTML-formatted vector
  HTML(cost_vec_html)
}

# Função para criar o output vetorx
output_vetorx <- function(input, num_variables, num_restrictions) {

    coefficients_z <- c(paste0("x", 1:(num_variables+num_restrictions)))
    #cost_vec_matricial <- matrix(coefficients_z, nrow = 1, ncol = length(coefficients_z), byrow = FALSE

  cost_vec_matricial_html <- paste0("<strong>vetor de variáveis é: </strong>", "$$x=$$",
                          withMathJax("\\begin{bmatrix}",
                                      paste(coefficients_z, collapse = " \\\\ "),
                                      "\\end{bmatrix}"))
  HTML(cost_vec_matricial_html)
}


#A Função para criar a matriz de coeficientes foi alterada e está no server
```

```
# Função para criar o vetor b


output_vetorb <- function(input, num_variables, num_restrictions) {
  coeficientes <- NULL

  for (i in 1:input$num_restrictions) {
    coeficientes <-c(coeficientes, as.numeric(unlist(strsplit(input[[paste0("R", i)]], ",")))[input$num_
  }
  b_matricial <- abs(coeficientes)

  b_matricial_html <- paste0("<strong>vetor de recursos é: </strong>", "$$b=$$",
                          withMathJax("\\begin{bmatrix}",
                                      paste(b_matricial, collapse = " \\\\ "),
                                      "\\end{bmatrix}"))

  HTML(b_matricial_html)
}

#função para verificar direção das restrições e retornar um aviso ao usuário
verificar_tipo_direcao <- function(input, num_restrictions) {
  tipos_direcao <- sapply(1:num_restrictions, function(i) {
    input[[paste0("dir_", i)]]
  })

  if (all(tipos_direcao == "<=")) {
    return("O modelo atual se resolve pelo SIMPLEX.")
  } else {
    return("O modelo atual se resolve pelo SIMPLEX DUAS FASES.")
  }
}

#função para calcular os índices do primeiro particionamento do Simplex
calcular_indices <- function(input) {
  num_variables <- input$num_variables
  num_restrictions <- input$num_restrictions

  # Verificar se todas as direções são do tipo <=
  tipos_direcao <- sapply(1:num_restrictions, function(i) {
    input[[paste0("dir_", i)]]
  })

  if (!all(tipos_direcao == "<=")) {
    # Caso não seja verdade, retorna NULL
    return(NULL)
  }

  # Se todas as direções são do tipo <=, calcular os vetores IN e IB
  IN <- 1:num_variables
  IB <- (num_variables + 1):(num_variables + num_restrictions)

  return(list(IN = IN, IB = IB))
}

#função para criar os elementos matriciais do simplex
#
#Matriz A
# Função para criar a matriz A com os coeficientes de entrada
gerar_matriz_A <- function(input, num_variables, num_restrictions) {
```

```
  num_variables <- input$num_variables
  num_restrictions <- input$num_restrictions

  # Inicializa a matriz com zeros
  A <- matrix(0, nrow = num_restrictions, ncol = num_variables)

  # Preenche a matriz com os coeficientes das restrições
  for (i in 1:num_restrictions) {
    restriction <- input[[paste0("R", i)]]
    linha<-as.numeric(unlist(strsplit(restriction, ",")))
    A[i, ] <- linha[1:num_variables]
  }
  A <- cbind(A,diag(num_restrictions))

  return(A)
}

#Vetor b
# Função para criar o vetor b com os coeficientes de entrada
gerar_b <- function(input, num_variables, num_restrictions) {
    coeficientes <- NULL

  for (i in 1:input$num_restrictions) {
    coeficientes <-c(coeficientes, as.numeric(unlist(strsplit(input[[paste0("R", i)]], ",")))[input$num_
  }
  b <- abs(coeficientes)

  return(b)
}


#Vetor custo
# Função para criar vetor custo
vetorcusto <- function(input, num_variables, num_restrictions) {
  num_variables <- input$num_variables
  num_restrictions <- input$num_restrictions
  coefficients_z <- as.numeric(unlist(strsplit(input[["cost_vec"]], ",")))
  # Modify coefficients and objective function based on input$obj
  if (input$obj == "max") {
    coefficients_z <- -coefficients_z
    coefficients_z <- c(coefficients_z, rep(0, num_restrictions))
    } else {
    coefficients_z <- c(coefficients_z, rep(0, num_restrictions))
    }
  return(coefficients_z)
}
#Solução Corrente
# Função gerar solução corrente
x_solucao<-function(input, num_variables, num_restrictions, IB=NULL){
  #num_variables <- input$num_variables
  #num_restrictions <- input$num_restrictions
  if(is.null(IB)) {IB=(input$num_variables+1):(input$num_restrictions+input$num_variables)}
I=1:(input$num_variables+input$num_restrictions)
IN=setdiff(I,IB)
A <- gerar_matriz_A(input, input$num_variables, input$num_restrictions)
B=A[,IB]
b<-gerar_b(input, input$num_variables, input$num_restrictions)
xB = solve(B)%*%b
rownames(xB)<-(paste0("x",IB))
```

```
x=rep(0,length(IN)+length(IB))
j=1
for (i in IB){
x[i]=xB[j]
j=j+1}
return(x)
}

# Função para criar o output output_x da iteração 1
output_x <- function(input, num_variables,num_restrictions) {
 x<-x_solucao(input, input$num_variables, input$num_restrictions)
 # Use HTML to format the matrix
  x_html <- paste0("<strong> </strong>", "$$x^T=$$",
                           withMathJax("\\begin{bmatrix}",
                                       paste(x, collapse = " & "),
                                       "\\end{bmatrix}"))

  # Return the HTML-formatted vector
  HTML(x_html)
}
#Vetor lambda
# Função para criar o vetor lambda para indices IB
lambda_vetor <- function(input, num_variables, num_restrictions, IB=NULL) {
  num_variables <- input$num_variables
  num_restrictions <- input$num_restrictions
  if(is.null(IB)) {IB=(num_variables+1):(num_restrictions+num_variables)}

  A <- gerar_matriz_A(input, num_variables, num_restrictions)
  custo <-vetorcusto(input, num_variables, num_restrictions)
  B=A[,IB]
  cB=custo[IB]
  lambda = solve(t(B))%*%cB
  return(lambda)
}

# Função para criar o output output_lambda da iteração 1
output_lambda <- function(input, num_variables, num_restrictions) {
 lambda<-t(lambda_vetor(input, num_variables, num_restrictions))
 # Use HTML to format the matrix
  lambda_html <- paste0("<strong> </strong>", "$$\\lambda^T=$$",
                           withMathJax("\\begin{bmatrix}",
                                       paste(lambda, collapse = " & "),
                                       "\\end{bmatrix}"))

  # Return the HTML-formatted vector
  HTML(lambda_html)
}

#Vetor custo relativo
# Função para criar o vetor custo relativo
CN_vetor <- function(input, num_variables, num_restrictions, IB=NULL) {
  num_variables <- input$num_variables
  num_restrictions <- input$num_restrictions
  if(is.null(IB)) {IB=(num_variables+1):(num_restrictions+num_variables)}
  IN<-1:num_variables

A <- gerar_matriz_A(input, input$num_variables, input$num_restrictions)
custo <-vetorcusto(input, input$num_variables, input$num_restrictions)
```

```
cB=custo[IB]
cN=custo[IN]
B=A[,IB]
lambda<-lambda_vetor(input, input$num_variables, input$num_restrictions)
cN_relativo=NULL
for (i in seq_along(cN))
cN_relativo[i] = cN[i] - t(lambda)%*%A[,IN[i]]


  return(cN_relativo)
}
# Função para criar o output cN_relativo da iteração 1
output_cN_relativo <- function(input, num_variables, num_restrictions) {

cN_relativo = CN_vetor(input, input$num_variables, input$num_restrictions)
# Use HTML to format the matrix
  cN_relativo_html <- paste0("<strong> </strong>", "$$c_N^{'T}=$$",
                          withMathJax("\\begin{bmatrix}",
                                      paste(cN_relativo, collapse = " & "),
                                      "\\end{bmatrix}"))

  # Return the HTML-formatted vector
  HTML(cN_relativo_html)
}
# Função para criar o vetor tamanho de passo
e_vetor <- function(input, num_variables, num_restrictions, IB=NULL) {
  #input$num_variables
 # input$num_restrictions
  if(is.null(IB)) {IB=(input$num_variables+1):(input$num_restrictions+input$num_variables)}
  IN<-1:input$num_variables
A <- gerar_matriz_A(input, input$num_variables, input$num_restrictions)
custo <- vetorcusto(input, input$num_variables, input$num_restrictions)
x<-x_solucao(input, input$num_variables, input$num_restrictions)
cN_relativo = CN_vetor(input, input$num_variables, input$num_restrictions)
cB=custo[IB]
cN=custo[IN]
B=A[,IB]

if(all(cN_relativo>=0)){e<-c("Vazio~ pois~ Solução~ corrente~ ótima")}else{
k=which.min(cN_relativo[cN_relativo<0])
y = (solve(B)%*%A[,IN[k]])[,1]
xB=x[IB]

e<-round(xB/y,4)
}
 return(e)
}
# Função para criar o output tamanho de passo da iteração 1
output_passo <- function(input, num_variables, num_restrictions){
e<- e_vetor(input, input$num_variables, input$num_restrictions) #lambda_vetor(input)#

# Use HTML to format the matrix
  passo_html <- paste0("<strong> </strong>", "$$\\epsilon=$$",
                          withMathJax("\\begin{bmatrix}",
                                      paste(e, collapse = " & "),
                                      "\\end{bmatrix}"))

  # Return the HTML-formatted vector
  HTML(passo_html)
```

```
}


ui <- fluidPage(
tags$script(src = "https://polyfill.io/v3/polyfill.min.js?features=es6", type = "text/javascript"),
tags$script(src = "https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.7/MathJax.js?config=TeX-MML-AM_CHT
tags$head(
  tags$style(HTML("
      @import url('https://fonts.googleapis.com/css2?family=Yusei+Magic&display=swap');
      body {
        background-color: #b3b3ff;
        color: black;
      }
      h2 {
        font-family: 'Yusei Magic', sans-serif;
      }
      .shiny-input-container {
        color: #474747;
      }"))
),
  titlePanel("Simplex-Formas"),
  fluidRow(column(12, h6("Desenvolvido por: Luciane Ferreira Alcoforado-AFA"))),
  sidebarLayout(
    sidebarPanel(
      tags$p(
    style = "color: red;",
    "Nota: Certifique-se de que a quantidade de valores informados entre vírgulas seja coerente com o nú
  ),
      sliderInput("num_variables", "Número de Variáveis:", min = 2, max = 5, value = 2),
      sliderInput("num_restrictions", "Número de Restrições:", min = 2, max = 6, value = 2),
      selectInput("obj", "Objetivo:", c("max", "min")),
      uiOutput("cost_vec_input"),
      uiOutput("restriction_inputs"),
      uiOutput("direction_inputs"),
      width = 3
    ),
    mainPanel(
      tabsetPanel(
        tabPanel("Forma Padrão",
          h3("Modelo"),
          uiOutput("output"),
          h3("Forma Padrão"),
          htmlOutput("output_formapadrao"), # Use htmlOutput para renderizar HTML diretamente
                  ),
        tabPanel("Forma Matricial",
          h3("Forma Padrão"),
          htmlOutput("output_formapadrao_matricial"), # Use htmlOutput para renderizar HTML diretamente
          h3("Elementos Matriciais da Forma Padrão"),
          HTML("<p> $$min: ~ c^Tx$$</p>"),
          HTML("<p> $$Sujeito ~ a: Ax=b$$</p>"),
          HTML("<p> $$x~\\geq~0; ~ b~\\geq~0$$</p>"),
          #HTML("<p>Expressão matemática: <span style='white-space: nowrap;'>$$Ax = b$$</span></p>"),
          #tags$script(HTML('MathJax.Hub.Config({tex2jax: {inlineMath: [["$$","$$"]]}});')),
          #tags$div(HTML("$$Ax = b$$")),
          #HTML("<p> $$x = \\begin{bmatrix} 1 \\\\ 1 \\\\ 1 \\end{bmatrix}$$ </p>"),
         #Inclua a expressão matemática usando withMathJax
          withMathJax(""),
          htmlOutput("output_vetorcusto"),  # Use htmlOutput para renderizar HTML diretamente
          htmlOutput("output_vetorx"),  # Use htmlOutput para renderizar HTML diretamente
```

```
          HTML("<strong> A matriz de coeficientes é:</strong>"),
         #htmlOutput("output_matriz_coeficientes"),
         htmlOutput("matriz_coeficientes1"),
           htmlOutput("output_vetorb")
         ),
        tabPanel("Particionamento",
         h3("Iteração 1 do Simplex"),
         HTML("<p> As restrições devem ser todas do tipo &le;. O modelo deve estar na forma padrão.</p>
         htmlOutput("tipo_modelo"),
         HTML("<p> </p>"),
         h3("Forma Padrão"),
         htmlOutput("output_formapadrao1"),
         h3("Particionamento Inicial"),
         HTML("<p> $$I_N ~e~ I_B: ~ contadores ~simplex$$</p>"),
         htmlOutput("vetores_indices"),
         HTML("<p> $$c^T = [c_N^T ~ c_B^T]: ~ vetor~ custo$$</p>"),
         htmlOutput("vetores_custos"),
         HTML("<p> $$x^T = [x_N^T ~ x_B^T]: ~ vetor~ das~ variáveis$$</p>"),
          htmlOutput("vetores_x"),
         HTML("<p> $$A = [N ~ B]: ~ matriz~dos~coeficientes$$</p>"),
         htmlOutput("matriz_coeficientes2"),
         htmlOutput("matriz_coeficientes3"),

         #Inclua a expressão matemática usando withMathJax
         withMathJax(""),

         ),
        tabPanel("Operações Iniciais",
         HTML("<p> $$I_N ~e~ I_B: ~ contadores ~simplex$$</p>"),
         htmlOutput("vetores_indices_1"),
         #HTML("<p> Observe os elementos matriciais do particionamento 1, identificando a variável não
         HTML("<p> $$Bx_B=b ~e~ B^{T}\\lambda = c_B$$</p>"),
         HTML("<p> $$x^T:~solução~corrente$$</p>"),
         htmlOutput("output_x"),
         HTML("<p> $$\\lambda^T:~multiplicador~simplex$$</p>"),
         htmlOutput("output_lambda"),
         HTML("<p> $$c'^{T}_{N}=c_N-\\lambda^TA | By=a_k, k \\in I_N$$</p>"),
         HTML("<p> $$\\epsilon = \\frac{x_z}{y_z}, z \\in I_B $$</p>"),
         HTML("<p> $$c'^{T}_{N}:~custo~relativo ~e~ \\epsilon:~tamanho~de~passo$$</p>"),




         htmlOutput("output_cN_relativo"),
         htmlOutput("output_passo")
        )

      )
    )
  )
)

server <- function(input, output, session) {
  # Create dynamic UI for cost vector based on user input
  output$cost_vec_input <- renderUI({
    num_variables <- input$num_variables
    textInput("cost_vec", "Vetor custo:", value = paste(rep("2", num_variables), collapse = ","))
  })
```

```
# Create dynamic UI for restrictions based on user input
output$restriction_inputs <- renderUI({
  num_restrictions <- input$num_restrictions
  num_variables <- input$num_variables
  restriction_list <- lapply(1:num_restrictions, function(i) {
    textInput(paste0("R", i), paste0("R", i, ":"), value <- paste(paste(rep("3", num_variables), collaps
  })
  do.call(tagList, restriction_list)
})




# Create dynamic UI for directions based on user input
  output$direction_inputs <- renderUI({
    num_restrictions <- input$num_restrictions
    direction_list <- lapply(1:num_restrictions, function(i) {
      selectInput(paste0("dir_", i), paste0("Direção ", i, ":"), c("<=", ">=", "="))
    })
    do.call(tagList, direction_list)
  })

 # Show the PPL model
  output$output <- renderUI({
    num_variables <- input$num_variables
    coefficients_z <- as.numeric(unlist(strsplit(input[["cost_vec"]], ",")))
    cost_vec_z <- paste0("z = ", paste(paste0(coefficients_z, sprintf("x%d", 1:num_variables)), collapse

    restrictions <- lapply(1:(input$num_restrictions), function(i) {
      # Convert comma-separated values to numeric vector
      coefficients <- as.numeric(unlist(strsplit(input[[paste0("R", i)]], ",")))
      paste0("R<sub>", i, "</sub>: ", paste(paste0(coefficients[1:num_variables], sprintf("x%d", 1:num_v
             " ", input[[paste0("dir_", i)]], " ", coefficients[num_variables + 1])
    })

    HTML(paste0("<p><strong>", input$obj, "</strong> ", cost_vec_z, "</p>",
                "<p><strong>sujeito a</strong></p>",
                paste(restrictions, collapse = "<br>"),  # Adicionar quebras de linha entre restrições
                "<p>", paste(sprintf("x%d  0", 1:num_variables), collapse = ", "), "</p>"))
  })

  # Renderizar o output para a Forma Padrão
  output$output_formapadrao <- renderText({
    output_formapadrao(input, input$num_variables, input$num_restrictions)
  })

  # Renderizar o output para a Forma Matricial
  output$output_formapadrao_matricial <- renderText({
    output_formapadrao(input, input$num_variables, input$num_restrictions)
  })

  # Renderizar o output para a Forma Matricial
  output$output_vetorcusto <- renderUI({
    output_vetorcusto(input, input$num_variables, input$num_restrictions)
  })
  # Renderizar o output para a Forma Matricial
  output$output_vetorx <- renderUI({
    output_vetorx(input, input$num_variables, input$num_restrictions)
```

```
  })

  # Renderizar o output para a Matriz de Coeficientes
  output$output_matriz_coeficientes <- renderUI({
    output_matriz_coeficientes(input, input$num_variables, input$num_restrictions)
  })

  # Renderizar o output para a Forma Matricial
  output$output_vetorb <- renderUI({
    output_vetorb(input, input$num_variables, input$num_restrictions)
  })

   # Função reativa para gerar a matriz A
  matriz_coeficientes_expr <- reactive({
    # Inicializa a string com a expressão da matriz
    expr <- "\\[ A = \\begin{bmatrix}"

    # Adicione o código da sua função output_matriz_coeficientes aqui
    # Início do código
    matriz_identidade <- diag(input$num_restrictions)
    for (i in 1:input$num_restrictions) {
      coefficients <- as.numeric(unlist(strsplit(input[[paste0("R", i)]], ",")))
      if (input[[paste0("dir_", i)]] == ">=") {
        matriz_identidade[i,] <- -matriz_identidade[i,]
      }
      coefficients_padrao <- c(coefficients[1:input$num_variables], matriz_identidade[i,], coefficients[
      if (coefficients_padrao[input$num_variables + input$num_restrictions + 1] < 0) {
        coefficients_padrao <- -coefficients_padrao
      }
      expr <- paste0(expr, paste(coefficients_padrao[1:(length(coefficients_padrao)-1)], collapse = " &
    }
    # Fim do código

    # Fecha a expressão da matriz
    expr <- paste0(expr, "\\end{bmatrix} \\]")
    return(expr)
  })


  # Use MathJax para renderizar a expressão da matriz
  output$matriz_coeficientes1 <- renderUI({
    withMathJax(matriz_coeficientes_expr())
  })



  #Outros outputs aba particionamento
# Renderizar o output para a Forma Padrão
  output$output_formapadrao1 <- renderText({
    output_formapadrao(input, input$num_variables, input$num_restrictions)
  })

  # Renderizar output indicando se o modelo usa SIMPLEX ou SIMPLEX DUAS FASES
  output$tipo_modelo <- renderText({
    verificar_tipo_direcao(input, input$num_restrictions)
  })
```

```
# Crie um output para os vetores IN e IB usando reatividade
  output$vetores_indices <- renderUI({
    # Teste se todos os input$dir_i são do tipo <=
    todos_menores_igual <- all(sapply(1:input$num_restrictions, function(i) input[[paste0("dir_", i)]] =

    # Se todos forem do tipo <=, crie os vetores IN e IB
    if (todos_menores_igual) {
      IN <- 1:input$num_variables
      IB <- (input$num_variables + 1):(input$num_restrictions + input$num_variables)

      # Crie o texto HTML para os vetores usando MathJax
      texto_html <- withMathJax(HTML(paste("<p> $$I_N =", toString(IN), "$$</p>",
                                           "<p> $$I_B =", toString(IB), "$$</p>")))
    } else {
      # Caso contrário, informe sobre o método simplex duas fases usando MathJax
      texto_html <- withMathJax(HTML("<p> O modelo atual se resolve pelo SIMPLEX DUAS FASES. </p>"))
    }

    return(texto_html)
  })

# Crie um output para os vetores C_N e C_B usando reatividade
  output$vetores_custos <- renderUI({
    # Teste se todos os input$dir_i são do tipo <=
    todos_menor_igual <- all(sapply(1:input$num_restrictions, function(i) input[[paste0("dir_", i)]] ==

    # Se todos forem do tipo <=, crie os vetores C_N e C_B
    if (todos_menor_igual) {
      IN <- 1:input$num_variables
      IB <- (input$num_variables + 1):(input$num_restrictions + input$num_variables)

      # Crie os vetores C_N e C_B
      C_N <- output_vetorcusto_N(input, input$num_variables, input$num_restrictions)
      C_B <- rep(0,length(IB))

      # Crie o texto HTML para os vetores usando MathJax
      texto_html <- withMathJax(HTML(paste("<p> ", toString(C_N), "</p>",
                                           "<p> $$c_B^T =[", toString(C_B), "]$$</p>")))
    } else {
      # Caso contrário, informe sobre o método simplex duas fases usando MathJax
      texto_html <- withMathJax(HTML("<p> O modelo atual se resolve pelo SIMPLEX DUAS FASES. </p>"))
    }

    return(texto_html)
  })

# Crie um output para os vetores x_N e x_B usando reatividade
  output$vetores_x <- renderUI({
    # Teste se todos os input$dir_i são do tipo <=
    todos_menor_igual <- all(sapply(1:input$num_restrictions, function(i) input[[paste0("dir_", i)]] ==

    # Se todos forem do tipo <=, crie os vetores x_N e x_B
    if (todos_menor_igual) {
      IN <- 1:input$num_variables
      IB <- (input$num_variables + 1):(input$num_restrictions + input$num_variables)

      # Crie os vetores x_N e x_B
      x_N <- paste0("x",IN)
      x_B <- paste0("x",IB)
```

```
      # Crie o texto HTML para os vetores usando MathJax
      texto_html <- withMathJax(HTML(paste("<p> $$x_N^T =[", toString(x_N), "]$$</p>",
                                           "<p> $$x_B^T =[", toString(x_B), "]$$</p>")))
    } else {
      # Caso contrário, informe sobre o método simplex duas fases usando MathJax
      texto_html <- withMathJax(HTML("<p> O modelo atual se resolve pelo SIMPLEX DUAS FASES. </p>"))
    }

    return(texto_html)
  })

# Função reativa para gerar a matriz N particionamento 1
  matriz_coeficientes_expr_N <- reactive({
    # Inicializa a string com a expressão da matriz
    expr <- "\\[ N = \\begin{bmatrix}"

    # Adicione o código da sua função output_matriz_coeficientes aqui
    # Início do código
    matriz_identidade <- diag(input$num_restrictions)
    for (i in 1:input$num_restrictions) {
      coefficients <- as.numeric(unlist(strsplit(input[[paste0("R", i)]], ",")))
      if (input[[paste0("dir_", i)]] == ">=") {
        matriz_identidade[i,] <- -matriz_identidade[i,]
      }
      coefficients_padrao <- c(coefficients[1:input$num_variables], matriz_identidade[i,], coefficients[
      if (coefficients_padrao[input$num_variables + input$num_restrictions + 1] < 0) {
        coefficients_padrao <- -coefficients_padrao
      }
      expr <- paste0(expr, paste(coefficients_padrao[1:(length(coefficients_padrao)-1-input$num_restrict
    }
    # Fim do código

    # Fecha a expressão da matriz
    expr <- paste0(expr, "\\end{bmatrix} \\]")
    return(expr)
  })

  # Use MathJax para renderizar a expressão da matriz
  output$matriz_coeficientes2 <- renderUI({
    withMathJax(matriz_coeficientes_expr_N())
  })

  # Função reativa para gerar a matriz B particionamento 1
  matriz_coeficientes_expr_B <- reactive({
    # Inicializa a string com a expressão da matriz
    expr <- "\\[ B = \\begin{bmatrix}"

    # Adicione o código da sua função output_matriz_coeficientes aqui
    # Início do código
    matriz_identidade <- diag(input$num_restrictions)
    for (i in 1:input$num_restrictions) {
      coefficients <- as.numeric(unlist(strsplit(input[[paste0("R", i)]], ",")))
      if (input[[paste0("dir_", i)]] == ">=") {
        matriz_identidade[i,] <- -matriz_identidade[i,]
      }
      coefficients_padrao <- c(coefficients[1:input$num_variables], matriz_identidade[i,], coefficients[
      if (coefficients_padrao[input$num_variables + input$num_restrictions + 1] < 0) {
        coefficients_padrao <- -coefficients_padrao
      }
```

```
      expr <- paste0(expr, paste(coefficients_padrao[(length(coefficients_padrao)-input$num_restrictions
    }
    # Fim do código

    # Fecha a expressão da matriz
    expr <- paste0(expr, "\\end{bmatrix} \\]")
    return(expr)
  })

  # Use MathJax para renderizar a expressão da matriz
  output$matriz_coeficientes3 <- renderUI({
    withMathJax(matriz_coeficientes_expr_B())
  })

# Operações Iniciais
#
#   #Outros outputs aba Operações Iniciais

 # Função reativa para gerar a matriz B particionamento 1
  matriz_coeficientes_expr_B <- reactive({
    # Inicializa a string com a expressão da matriz
    expr <- "\\[ B = \\begin{bmatrix}"

    # Adicione o código da sua função output_matriz_coeficientes aqui
    # Início do código
    matriz_identidade <- diag(input$num_restrictions)
    for (i in 1:input$num_restrictions) {
      coefficients <- as.numeric(unlist(strsplit(input[[paste0("R", i)]], ",")))
      if (input[[paste0("dir_", i)]] == ">=") {
        matriz_identidade[i,] <- -matriz_identidade[i,]
      }
      coefficients_padrao <- c(coefficients[1:input$num_variables], matriz_identidade[i,], coefficients[
      if (coefficients_padrao[input$num_variables + input$num_restrictions + 1] < 0) {
        coefficients_padrao <- -coefficients_padrao
      }
      expr <- paste0(expr, paste(coefficients_padrao[(length(coefficients_padrao)-input$num_restrictions
    }
    # Fim do código

    # Fecha a expressão da matriz
    expr <- paste0(expr, "\\end{bmatrix} \\]")
    return(expr)
  })

  # Use MathJax para renderizar a expressão da matriz
  output$matriz_coeficientes3 <- renderUI({
    withMathJax(matriz_coeficientes_expr_B())
  })


# Crie um output para os vetores IN e IB usando reatividade
  output$vetores_indices_1 <- renderUI({
    # Teste se todos os input$dir_i são do tipo <=
    todos_menores_igual <- all(sapply(1:input$num_restrictions, function(i) input[[paste0("dir_", i)]] =

    # Se todos forem do tipo <=, crie os vetores IN e IB
    if (todos_menores_igual) {
      IN <- 1:input$num_variables
      IB <- (input$num_variables + 1):(input$num_restrictions + input$num_variables)
```

```
      # Crie o texto HTML para os vetores usando MathJax
      texto_html <- withMathJax(HTML(paste("<p> $$I_N =", toString(IN), "$$</p>",
                                           "<p> $$I_B =", toString(IB), "$$</p>")))
    } else {
      # Caso contrário, informe sobre o método simplex duas fases usando MathJax
      texto_html <- withMathJax(HTML("<p> O modelo atual se resolve pelo SIMPLEX DUAS FASES. </p>"))
    }

    return(texto_html)
  })

  # Renderizar o output para a Forma Matricial do vetor custo relativo
  output$output_cN_relativo <- renderUI({
    # Teste se todos os input$dir_i são do tipo <=
    todos_menores_igual <- all(sapply(1:input$num_restrictions, function(i) input[[paste0("dir_", i)]] =

    # Se todos forem do tipo <=, crie os vetores IN e IB
    if (todos_menores_igual) {
      output_cN_relativo(input, input$num_variables, input$num_restrictions)
    } else {
      # Caso contrário, informe sobre o método simplex duas fases usando MathJax
      texto_html <- withMathJax(HTML("<p>  </p>"))
      return(texto_html)
    }

  })

  # Renderizar o output
  output$output_lambda <- renderUI({

    # Teste se todos os input$dir_i são do tipo <=
    todos_menores_igual <- all(sapply(1:input$num_restrictions, function(i) input[[paste0("dir_", i)]] =

    # Se todos forem do tipo <=, crie os vetores IN e IB
    if (todos_menores_igual) {
      output_lambda(input, input$num_variables, input$num_restrictions)
    } else {
      # Caso contrário, informe sobre o método simplex duas fases usando MathJax
      texto_html <- withMathJax(HTML("<p>  </p>"))
      return(texto_html)
      }
  })

# Renderizar o output
  output$output_x <- renderUI({
    # Teste se todos os input$dir_i são do tipo <=
    todos_menores_igual <- all(sapply(1:input$num_restrictions, function(i) input[[paste0("dir_", i)]] =

    # Se todos forem do tipo <=, crie os vetores IN e IB
    if (todos_menores_igual) {
      output_x(input, input$num_variables, input$num_restrictions)
    } else {
      # Caso contrário, informe sobre o método simplex duas fases usando MathJax
      texto_html <- withMathJax(HTML("<p> Observe que a direção de pelo menos uma restrição não é do tip
      return(texto_html)
    }

  })
```

```
  # Renderizar o output
  output$output_passo <- renderUI({
    # Teste se todos os input$dir_i são do tipo <=
    todos_menores_igual <- all(sapply(1:input$num_restrictions, function(i) input[[paste0("dir_", i)]] =

    # Se todos forem do tipo <=, crie os vetores IN e IB
    if (todos_menores_igual) {
      output_passo(input, input$num_variables, input$num_restrictions)
    } else {
      # Caso contrário, informe sobre o método simplex duas fases usando MathJax
      texto_html <- withMathJax(HTML("<p>   </p>"))
      return(texto_html)
    }

  })




}

shinyApp(ui, server)
```

Comentários Técnicos

1. Interface do Usuário (UI)

- **Inputs Dinâmicos**: O aplicativo permite a entrada de dados flexíveis, como o número de variáveis e restrições. O `sliderInput` e `selectInput` oferecem ao usuário a capacidade de ajustar essas entradas de forma intuitiva.
- **Validação do Input**: Existe uma nota no painel lateral que alerta o usuário sobre a consistência dos dados fornecidos. Esta é uma boa prática para evitar erros na execução do Simplex.
- **Uso de MathJax**: A renderização de fórmulas matemáticas é realizada por meio do MathJax, o que melhora a legibilidade das expressões, permitindo a apresentação adequada de notações matriciais e outras fórmulas.
- **Outputs Customizados**: O aplicativo usa funções customizadas (`output_vetorcusto`, `output_formapadrao`, etc.) para exibir as diferentes etapas do Simplex, como a forma padrão, o vetor de custos e a matriz de coeficientes.

2. Lógica do Servidor

- **Manipulação Dinâmica dos Coeficientes**: O aplicativo processa os dados de entrada (como os coeficientes da função objetivo e as restrições) e os converte em vetores e matrizes, que são exibidos na interface. A lógica é adaptada tanto para maximização quanto para minimização, com inversão dos sinais quando apropriado.
- **Verificação do Tipo de Restrições**: Existe uma verificação para determinar o método adequado a ser utilizado (Simplex padrão ou Duas Fases) com base na direção das restrições (`<=` ou `>=`). Esse tipo de checagem automatiza a escolha do algoritmo correto.
- **Particionamento para o Simplex**: O código inclui a lógica de particionamento inicial (`IN` e `IB`) para preparar os índices das variáveis básicas e não-básicas, um passo fundamental no algoritmo Simplex.
- **Operações Iniciais**: O código inclui cálculos matriciais referentes a primeira iteração do algoritmo Simplex. Esta funcionalidade foi atualizada na versão 1.0.1 e não estava contemplada na versão 1.0.0.

- **Atualização Dinâmica**: O uso de `uiOutput` e `htmlOutput` para renderizar conteúdo dinâmico permite que o aplicativo seja responsivo às mudanças nas entradas sem necessidade de recarregar a página.

3. Fatores Positivos

- **Flexibilidade e Usabilidade**: A interface é bastante interativa, com sliders e campos de texto para ajustar as variáveis e coeficientes. A lógica de verificação de inconsistências ajuda a evitar entradas incorretas.
- **Clareza Visual**: A integração com o MathJax melhora significativamente a apresentação das fórmulas, algo crucial para problemas matemáticos complexos como o Simplex.
- **Organização por Abas**: O uso de abas para separar os diferentes estágios do Simplex (Forma Padrão, Forma Matricial, Particionamento e Operações Iniciais) facilita a navegação e organiza as informações de maneira lógica.

4. Melhorias Implementadas

- **Tratamento de Erros**: Correção na lógica das funções que calculam as operações matriciais.
- **Feedback Visual**: Inclusão de feedback visual avisando o usuário quando uma entrada estiver inválida ou quando o método Simplex não puder ser resolvido com os dados fornecidos.
- **Exibição de Fórmulas e Resultados**: Inclusão de nova aba contendo fórmulas e resultados de operações matriciais, permitindo que o usuário possa realizar exercícios diversos, tendo o aplicativo como um gabarito em tempo real.