Apêndice H

Aqui apresentamos os códigos das duas versões do aplicativo Iterações, inicialmente denominado Entra e Sai da Base.

Código do aplicativo Simplex Entra e Sai da Base v.1.0.0

Data de Publicação: 04 de agosto de 2024

```
library(shiny)

# Versão anterior a app0
## Trabalhando para melhorar a partir desta versão
ui <- fluidPage(
  titlePanel("Simplex-Variável que entra e que sai da Base."),
  tags$div(style = "font-size: small; margin-bottom: 20px;",
           "Este aplicativo auxilia nos cálculos do custo relativo e tamanho de passo, considerando um p
  ),
  tags$div(style = "font-size: small; margin-bottom: 20px;",
           "Autoria: Luciane Ferreira Alcoforado - AFA"
  ),

  sidebarLayout(
    sidebarPanel(
      fluidRow(
        column(6, numericInput("n", "Variáveis (n):", min = 1, value = 2)),
        column(6, numericInput("m", "Restrições (m):", min = 2, value = 3))
      ),
      uiOutput("input_ui"),
      verbatimTextOutput("error_message")
    ),

    mainPanel(
      tabsetPanel(
        tabPanel("Vetores de Índices", uiOutput("vetores_indices")),
        tabPanel("IB", verbatimTextOutput("output_IB")),
        tabPanel("IN", verbatimTextOutput("output_IN")),
        tabPanel("c", verbatimTextOutput("output_c")),
        tabPanel("A", verbatimTextOutput("output_A")),
        tabPanel("B", verbatimTextOutput("output_B")),
        tabPanel("b", verbatimTextOutput("output_b")),

        tabPanel("Custo Relativo", verbatimTextOutput("output_CR")),
        tabPanel("Tamanho de Passo", verbatimTextOutput("output_E")),
        tabPanel("Solução Corrente", verbatimTextOutput("output_x")),

        tabPanel("Próxima Iteração", htmlOutput("output_proxima_iteracao"))
      )
    )
  )
)

server <- function(input, output, session) {

  values <- reactiveValues(
    c = NULL,
    A = NULL,
    dir = NULL,
    b = NULL,
```

```
    IB = NULL,
    IN = NULL,
    I = NULL,
    error_message = NULL,
    k = NULL,
    s = NULL
  )

  observe({
    n <- input$n
    m <- input$m

    output$input_ui <- renderUI({
      tagList(

        p("Analise o 'Custo Relativo' para decidir o IN que deve entrar na base e o 'tamanho de passo'
          para alterar o índice IB que deve sair."),
        textInput("input_IB", "Índice das Variáveis Básicas - IB (modifique os valores para outras itera
                  value = paste(seq.int(n + 1, length.out = m), collapse = ","),
                  placeholder = paste(seq.int(n + 1, length.out = m), collapse = ",")),
        textInput("input_c", "Coeficientes da Função Objetivo (separados por vírgula):",
                  value = paste(sample(1:20*-1, n, replace = TRUE), collapse = ","),
                  placeholder = paste(sample(1:20*-1, n, replace = TRUE), collapse = ",")),

        textInput("input_A", "Coeficientes das Restrições (separar valores por , na mesma linha e ; entr
                  value = paste(rep(paste(rep(1, n), collapse = ","), m), collapse = ";"),
                  placeholder = paste(rep(paste(rep(1, n), collapse = ","), m), collapse = ";")),

        textInput("input_b", "Limites das Restrições (separados por vírgula):",
                  value = paste(sample(c(1:5, 8), m, replace = TRUE), collapse = ","),
                  placeholder = paste(sample(c(1:5, 8), m, replace = TRUE), collapse = ","))

      )
    })

    observe({
      c_input <- input$input_c
      A_input <- input$input_A
      dir_input <- input$input_dir
      b_input <- input$input_b
      IB_input <- input$input_IB

      tryCatch({
        if (nzchar(c_input) && nzchar(A_input) && nzchar(b_input) && nzchar(IB_input)) {
          c_vals <- c(as.numeric(strsplit(c_input, ",")[[1]]), rep(0, m))
          A_vals <- as.numeric(unlist(strsplit(gsub(" ", "", A_input), "[,;]")))
          b_vals <- as.numeric(strsplit(b_input, ",")[[1]])
          IB_vals <- as.numeric(strsplit(IB_input, ",")[[1]])
          I <- 1:(n + m)
          IN_vals <- setdiff(I, IB_vals)

          if (length(A_vals) == m * n) {
            # Reshape A_vals para uma matriz m x n
            A_matrix <- matrix(A_vals, nrow = m, byrow = TRUE)

            # Adicionar a matriz diagonal à matriz A
            A_matrix_diagonal <- cbind(A_matrix, diag(m))

            B <- A_matrix_diagonal[, IB_vals, drop = FALSE]
```

```
        # Verificar se o determinante de B é dif zero
        if (det(B) != 0) {

        values$B <- B
        N <- A_matrix_diagonal[, IN_vals, drop = FALSE]
        xB <- solve(B) %*% b_vals

        rownames(xB) <- paste0("x", IB_vals)
        x <- rep(0, length(IN_vals) + length(IB_vals))
        names(x) <- c(paste0("x", IN_vals), paste0("x", IB_vals))

        # Atualizar x com valores de xB
        x[names(x) %in% rownames(xB)] <- xB

        cB <- c_vals[IB_vals]
        cN <- c_vals[IN_vals]
        z=sum(xB*cB)
        xz<- c(x,z=z)
        lambda <- solve(t(B)) %*% cB

        cN_relativo <- numeric(length(cN))
        for (i in seq_along(cN)) {
          cN_relativo[i] <- cN[i] - t(lambda) %*% A_matrix_diagonal[, IN_vals[i]]
        }

        if (all(cN_relativo >= 0)) {
          e <- "Atenção, os custos relativos são todos positivos => sol. ótima!"
          values$x <- x
          values$xz <- xz
          values$CR <- cN_relativo
          values$E <- e
          values$k <- NULL
          values$s <- NULL
        } else {
          k <- which.min(cN_relativo[cN_relativo < 0])
          y <- solve(B) %*% A_matrix_diagonal[, IN_vals[k]]
          e <- round(xB / y, 4)
          values$k <- IN_vals[k] #atentar k é posição e não o índice da posição.

          # Calcular a posição do menor valor de e que seja positivo e diferente de Inf
          valid_e <- min(e[!is.infinite(e) & e > 0])
          if (length(valid_e) > 0) {
            s <- IB_vals[which(e == valid_e)][1]
            values$E <- e
            values$k <- IN_vals[k] #atentar k é posição e não o índice da posição.
            values$s <- s
          } else {
            s <- NA
            k <- which.min(cN_relativo[cN_relativo < 0])
            e <- "Não há solução ótima"
            values$E <- e
            values$k <- IN_vals[k] #atentar k é posição e não o índice da posição.
            values$s <- s
          }
        }

        # Validar input_IB
        if (length(IB_vals) != m || any(!IB_vals %in% 1:(n + m)) || length(unique(IB_vals)) != m) {
          values$error_message <- "IB deve conter exatamente m valores únicos, inteiros, entre 1 e n
```

```
            values$IB <- NULL
            values$IN <- NULL
          } else {
            values$error_message <- NULL

            if (length(c_vals) == (n + m)) {
              values$c <- c_vals
            }
            if (ncol(A_matrix_diagonal) == (n + m)) {
              values$A <- A_matrix_diagonal
            }

            if (ncol(B) == (m)) {
              values$B <- B
            }

            if (length(b_vals) == m) {
              values$b <- b_vals
            }
            if (length(IB_vals) == m) {
              values$IB <- IB_vals
              values$IN <- IN_vals
              values$CR <- cN_relativo
              values$E <- e
              values$E_pos <- e
              values$s <- s
              values$k <- IN_vals[k] #atentar k é posição e não o índice da posição.
              values$x <- x
              values$xz <- xz
            }
          }
        } else {
            values$error_message <- "A matriz B é singular (det(B) = 0). O modelo precisa de revisão c
            values$B <- B
            values$k <- NULL
            values$s <- NULL
            values$IB <- IB_vals
            values$IN <- IN_vals
            values$xz <- "Rever o Particionamento ou as restrições"
            values$E <- "Rever o Particionamento ou as restrições"
            values$CR <- "Rever o Particionamento ou as restrições"
            return()
          }
          } else {
          values$error_message <- "Número incorreto de coeficientes para A."
        }
      } else {
        values$error_message <- "Certifique-se de que todos os campos estejam preenchidos corretamente
      }
    }, error = function(e) {
      values$error_message <- paste("Erro no processamento:", e$message)
    })
  })

  output$output_c <- renderPrint({
    if (!is.null(values$c)) values$c else "Nenhum valor para c"
  })

  output$output_A <- renderPrint({
```

```
      if (!is.null(values$A)) values$A else "Nenhum valor para A"
    })

    output$output_B <- renderPrint({
      if (!is.null(values$B)) values$B else "Nenhum valor para B"
    })

    output$output_b <- renderPrint({
      if (!is.null(values$b)) values$b else "Nenhum valor para b"
    })

    output$output_IB <- renderPrint({
      if (!is.null(values$IB)) values$IB else "Nenhum valor para IB"
    })

    output$output_IN <- renderPrint({
      if (!is.null(values$IN)) values$IN else "Nenhum valor para IN"
    })

    output$error_message <- renderText({
      if (!is.null(values$error_message)) values$error_message else ""
    })

    output$output_CR <- renderPrint({
      if (!is.null(values$CR)) values$CR else "Nenhum valor para Custo Relativo"
    })

    output$output_E <- renderPrint({
      if (!is.null(values$E)) values$E else "Nenhum valor para Tamanho de Passo"
    })

    output$output_x <- renderPrint({
      if (all(values$x >= 0)) values$xz else "Reveja seu particionamento pois a solução corrente é inviá
    })

    output$output_proxima_iteracao <- renderText({
      if (!is.null(values$k) && !is.null(values$s)) {
        paste("Índice da variável que deve entrar na base (k):", values$k,
              "<br><br>Índice da variável que deve sair da base (s):", values$s)
      } else {
        "Não é possível determinar a próxima iteração"
      }
    })

    output$vetores_indices <- renderUI({
      if (!is.null(values$IN) && !is.null(values$IB)) {
        withMathJax(HTML(paste(
          "<p>Índice das variáveis Não Básicas:\n</p>",
          "<p>$$I_N = \\{", paste(values$IN, collapse = ", "), "\\}$$</p>",
          "<p>Índice das variáveis Básicas:\n</p>",
          "<p>$$I_B = \\{", paste(values$IB, collapse = ", "), "\\}$$</p>"

        )))
      } else {
        "Nenhum valor para IN ou IB"
      }
    })
  })
}
```

```
shinyApp(ui, server)
```

Código do aplicativo Simplex Iterações v.1.1.0

Data de Publicação: 06 de março de 2025

Após ajustes de layout a versão final possue o seguinte código:

```
library(shiny)
```

```
Warning: pacote 'shiny' foi compilado no R versão 4.4.2
```

```
#versao v1.1.0

# Versão anterior a app2 (versão que foi publicada em agosto de 2024)
## Trabalhando para melhorar a partir desta versão
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_C
  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 Journey - Iterações"),
  tags$div(style = "font-size: small; margin-bottom: 20px;",
           "Este aplicativo auxilia nos cálculos do custo relativo e tamanho de passo, considerando um p
  ),
  tags$div(style = "font-size: small; margin-bottom: 20px;",
           "Autoria: Luciane Ferreira Alcoforado - AFA \\ v1.1.0"
  ),

  sidebarLayout(
    sidebarPanel(
      fluidRow(
        column(6, numericInput("n", "Variáveis (n):", min = 1, value = 2)),
        column(6, numericInput("m", "Restrições (m):", min = 2, value = 3))
      ),
      uiOutput("input_ui"),
      verbatimTextOutput("error_message")
    ),

    mainPanel(
      tabsetPanel(
        tabPanel("Vetores de Índices", uiOutput("vetores_indices")),
        tabPanel("IB", verbatimTextOutput("output_IB")),
        tabPanel("IN", verbatimTextOutput("output_IN")),
        tabPanel("c", verbatimTextOutput("output_c")),
```

```
        tabPanel("A", verbatimTextOutput("output_A")),
        tabPanel("B", verbatimTextOutput("output_B")),
        tabPanel("b", verbatimTextOutput("output_b")),

        tabPanel("Custo Relativo", verbatimTextOutput("output_CR")),
        tabPanel("Tamanho de Passo", verbatimTextOutput("output_E")),
        tabPanel("Solução Corrente", uiOutput("output_x")),

        tabPanel("Próxima Iteração", htmlOutput("output_proxima_iteracao")),
        tabPanel("Consulte as Fórmulas", uiOutput("output_consulte_formulas"))
      )
    )
  )
)

server <- function(input, output, session) {

  values <- reactiveValues(
    c = NULL,
    A = NULL,
    dir = NULL,
    b = NULL,
    IB = NULL,
    IN = NULL,
    I = NULL,
    error_message = NULL,
    k = NULL,
    s = NULL
  )

  observe({
    n <- input$n
    m <- input$m

    output$input_ui <- renderUI({
      tagList(

        p("Analise o 'Custo Relativo' para decidir o IN que deve entrar na base e o 'tamanho de passo'
          para alterar o índice IB que deve sair."),
        textInput("input_IB", "Índice das Variáveis Básicas - IB (modifique os valores para outras itera
                  value = paste(seq.int(n + 1, length.out = m), collapse = ","),
                  placeholder = paste(seq.int(n + 1, length.out = m), collapse = ",")),
        textInput("input_c", "Coeficientes da Função Objetivo (separados por vírgula):",
                  value = paste(sample(1:20*-1, n, replace = TRUE), collapse = ","),
                  placeholder = paste(sample(1:20*-1, n, replace = TRUE), collapse = ",")),

        textInput("input_A", "Coeficientes das Restrições (separar valores por , na mesma linha e ; entr
                  value = paste(rep(paste(rep(1, n), collapse = ","), m), collapse = ";"),
                  placeholder = paste(rep(paste(rep(1, n), collapse = ","), m), collapse = ";")),

        textInput("input_b", "Limites das Restrições (separados por vírgula):",
                  value = paste(sample(c(1:5, 8), m, replace = TRUE), collapse = ","),
                  placeholder = paste(sample(c(1:5, 8), m, replace = TRUE), collapse = ","))

      )
    })

    observe({
      c_input <- input$input_c
```

```
A_input <- input$input_A
dir_input <- input$input_dir
b_input <- input$input_b
IB_input <- input$input_IB

tryCatch({
  if (nzchar(c_input) && nzchar(A_input) && nzchar(b_input) && nzchar(IB_input)) {
    c_vals <- c(as.numeric(strsplit(c_input, ",")[[1]]), rep(0, m))
    A_vals <- as.numeric(unlist(strsplit(gsub(" ", "", A_input), "[,;]")))
    b_vals <- as.numeric(strsplit(b_input, ",")[[1]])
    IB_vals <- as.numeric(strsplit(IB_input, ",")[[1]])
    I <- 1:(n + m)
    IN_vals <- setdiff(I, IB_vals)

    if (length(A_vals) == m * n) {
      # Reshape A_vals para uma matriz m x n
      A_matrix <- matrix(A_vals, nrow = m, byrow = TRUE)

      # Adicionar a matriz diagonal à matriz A
      A_matrix_diagonal <- cbind(A_matrix, diag(m))

      B <- A_matrix_diagonal[, IB_vals, drop = FALSE]

      # Verificar se o determinante de B é dif zero
      if (det(B) != 0) {

      values$B <- B
      N <- A_matrix_diagonal[, IN_vals, drop = FALSE]
      xB <- solve(B) %*% b_vals

      rownames(xB) <- paste0("x", IB_vals)
      x <- rep(0, length(IN_vals) + length(IB_vals))
      names(x) <- c(paste0("x", IN_vals), paste0("x", IB_vals))

      # Atualizar x com valores de xB
      x[names(x) %in% rownames(xB)] <- xB

      cB <- c_vals[IB_vals]
      cN <- c_vals[IN_vals]
      z=sum(xB*cB)
      xz<- c(x,z=z)
      lambda <- solve(t(B)) %*% cB

      cN_relativo <- numeric(length(cN))
      for (i in seq_along(cN)) {
        cN_relativo[i] <- cN[i] - t(lambda) %*% A_matrix_diagonal[, IN_vals[i]]
      }
      cN_relativo<-as.matrix(cN_relativo)
      rownames(cN_relativo)<-paste0("c",IN_vals)

      if (all(cN_relativo >= 0)) {
        e <- "Atenção, os custos relativos são todos positivos => sol. ótima!"
        values$x <- x
        values$xz <- xz
        values$CR <- cN_relativo
        values$E <- e
        values$k <- NULL
        values$s <- NULL
      } else {
```

```
            k <- which.min(cN_relativo[cN_relativo < 0])
            y <- solve(B) %*% A_matrix_diagonal[, IN_vals[k]]
            e <- round(xB / y, 4)
            values$k <- IN_vals[k] #atentar k é posição e não o índice da posição.

            # Calcular a posição do menor valor de e que seja positivo e diferente de Inf
            valid_e <- min(e[!is.infinite(e) & e > 0])
            if (length(valid_e) > 0) {
              s <- IB_vals[which(e == valid_e)][1]
              values$E <- e
              values$k <- IN_vals[k] #atentar k é posição e não o índice da posição.
              values$s <- s
            } else {
              s <- NA
              k <- which.min(cN_relativo[cN_relativo < 0])
              e <- "Não há solução ótima"
              values$E <- e
              values$k <- IN_vals[k] #atentar k é posição e não o índice da posição.
              values$s <- s
            }
          }

          # Validar input_IB
          if (length(IB_vals) != m || any(!IB_vals %in% 1:(n + m)) || length(unique(IB_vals)) != m) {
            values$error_message <- "IB deve conter exatamente m valores únicos, inteiros, entre 1 e n
            values$IB <- NULL
            values$IN <- NULL
          } else {
            values$error_message <- NULL

            if (length(c_vals) == (n + m)) {
              values$c <- c_vals
            }
            if (ncol(A_matrix_diagonal) == (n + m)) {
              values$A <- A_matrix_diagonal
            }

            if (ncol(B) == (m)) {
              values$B <- B
            }

            if (length(b_vals) == m) {
              values$b <- b_vals
            }
            if (length(IB_vals) == m) {
              values$IB <- IB_vals
              values$IN <- IN_vals
              values$CR <- cN_relativo
              values$E <- e
              values$E_pos <- e
              values$s <- s
              values$k <- IN_vals[k] #atentar k é posição e não o índice da posição.
              values$x <- x
              values$xz <- xz
            }
          }
        } else {
            values$error_message <- "A matriz B é singular (det(B) = 0). O modelo precisa de revisão c
            values$B <- B
```

```
              values$k <- NULL
              values$s <- NULL
              values$IB <- IB_vals
              values$IN <- IN_vals
              values$xz <- "Rever o Particionamento ou as restrições"
              values$E <- "Rever o Particionamento ou as restrições"
              values$CR <- "Rever o Particionamento ou as restrições"
              return()
            }
            } else {
            values$error_message <- "Número incorreto de coeficientes para A."
          }
        } else {
          values$error_message <- "Certifique-se de que todos os campos estejam preenchidos corretamente
        }
      }, error = function(e) {
        values$error_message <- paste("Erro no processamento:", e$message)
      })
    })

    output$output_c <- renderPrint({
      if (!is.null(values$c)) values$c else "Nenhum valor para c"
    })

    output$output_A <- renderPrint({
      if (!is.null(values$A)) values$A else "Nenhum valor para A"
    })

    output$output_B <- renderPrint({
      if (!is.null(values$B)) values$B else "Nenhum valor para B"
    })

    output$output_b <- renderPrint({
      if (!is.null(values$b)) values$b else "Nenhum valor para b"
    })

    output$output_IB <- renderPrint({
      if (!is.null(values$IB)) values$IB else "Nenhum valor para IB"
    })

    output$output_IN <- renderPrint({
      if (!is.null(values$IN)) values$IN else "Nenhum valor para IN"
    })

    output$error_message <- renderText({
      if (!is.null(values$error_message)) values$error_message else ""
    })

    output$output_CR <- renderPrint({
      if (!is.null(values$CR)) values$CR else "Nenhum valor para Custo Relativo"
    })

    output$output_E <- renderPrint({
      if (!is.null(values$E)) values$E else "Nenhum valor para Tamanho de Passo"
    })

##    output$output_x <- renderPrint({
 #     if (all(values$x >= 0)) values$xz else "Reveja seu particionamento pois a solução corrente é invi
 #   })
```

```
    output$output_x <- renderUI({
      if (all(values$x >= 0)) {
        values$I=c(values$IN,values$IB)
        withMathJax(HTML(paste0(
          "<p>\\(x_",values$I, "=", values$x, "\\)</p>"
        )))
      } else {
        HTML("<p>Reveja seu particionamento pois a solução corrente é inviável</p>")
      }
    })

    output$output_proxima_iteracao <- renderText({
      if (!is.null(values$k) && !is.null(values$s)) {
        paste("Índice da variável que deve entrar na base (k):", values$k,
              "<br><br>Índice da variável que deve sair da base (s):", values$s)
      } else {
        "Não é possível determinar a próxima iteração"
      }
    })

    output$vetores_indices <- renderUI({
      if (!is.null(values$IN) && !is.null(values$IB)) {
        withMathJax(HTML(paste(
          "<p>Índice das variáveis Não Básicas:\n</p>",
          "<p>$$I_N = \\{", paste(values$IN, collapse = ", "), "\\}$$</p>",
          "<p>Índice das variáveis Básicas:\n</p>",
          "<p>$$I_B = \\{", paste(values$IB, collapse = ", "), "\\}$$</p>"

        )))
      } else {
        "Nenhum valor para IN ou IB"
      }
    })

    output$output_consulte_formulas <- renderUI({
      withMathJax(HTML("
      <h1>Formulário Método Simplex</h1>
      <p>Solução Básica Corrente <span>\\(x_B\\)</span>:</p>
      $$Bx_B=b$$
      <p>Solução Não Básica é sempre nula: <span>\\(x_N=0\\)</span>:</p>
      <p>Lambda <span>\\(\\lambda\\)</span>:</p>
      $$B^T \\lambda = c_B$$
      <p>Custo Relativo <span>\\(c^´_N\\)</span>:</p>
      $$c'_{i}=c_{i}-\\lambda^T \\cdot a_{i}, i\\in I_N$$
      <p>Direção Simplex <span>\\(y\\)</span>:</p>
      $$By=a_k$$
      <p>Tamanho de Passo <span>\\(\\epsilon\\)</span>:</p>
      $$\\epsilon_z=x_{z}/y_z, z \\in I_B$$


    "))
    })
  })
}

shinyApp(ui, server)
```

```
PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, please ma
```