Análise Numérica

Tarefa 7

Luiz Henrique Barretta Francisco - 202100155302

outubro/2025

1. Usando alguma linguagem de programação, escreva um programa para o método de Newton e para o método de Newton Modificado.

Funções Auxiliares

```
solve_gauss_pivot <- function(A, b) {
  n <- nrow(A)
  Ab <- cbind(A, b)

  # Eliminação
  for (k in 1:(n - 1)) {
    # Pivoteamento Parcia
    pivo_linha <- which.max(abs(Ab[k:n, k])) + k - 1

    # Troca as linhas
    if (pivo_linha != k) {
      temp <- Ab[k, ]
      Ab[k, ] <- Ab[pivo_linha, ]
      Ab[pivo_linha, ] <- temp}

    # Verifica matriz singular
    if (abs(Ab[k, k]) < 1e-10) {
      stop("Matriz singular detectada. O sistema pode não ter solução única.")}

    # Eliminação linhas abaixo
    for (i in (k + 1):n) {
      m <- Ab[i, k] / Ab[k, k]
      Ab[i, ] <- Ab[i, ] - m * Ab[k, ]}}

  # Retrosubstituição
  s <- numeric(n)
  for (i in n:1) {
    soma_produtos <- 0
    if (i < n) {
      soma_produtos <- sum(Ab[i, (i + 1):n] * s[(i + 1):n])}
    s[i] <- (Ab[i, n + 1] - soma_produtos) / Ab[i, i]}

  return(s)}

# Decomposição LU
lu_decompose <- function(A) {
  n <- nrow(A)
  p <- 1:n

  for (k in 1:(n - 1)) {
    # Pivoteamento
    pivo_linha <- which.max(abs(A[k:n, k])) + k - 1
    if (pivo_linha != k) {
      # Troca linhas
      tempA <- A[k, ]
      A[k, ] <- A[pivo_linha, ]
      A[pivo_linha, ] <- tempA
      temp_p <- p[k]
```

```
      p[k] <- p[pivo_linha]
      p[pivo_linha] <- temp_p}

    if (abs(A[k, k]) < 1e-10) {
      stop("Matriz singular encontrada durante a decomposição LU.")}

    # Fatoração
    for (i in (k + 1):n) {
      A[i, k] <- A[i, k] / A[k, k]
      for (j in (k + 1):n) {
        A[i, j] <- A[i, j] - A[i, k] * A[k, j]}}}
  return(list(LU = A, p = p))}

# Resolver Ax=b
lu_solve <- function(LU_decomp, b) {
  LU <- LU_decomp$LU
  p <- LU_decomp$p
  n <- nrow(LU)

  b <- b[p]
  y <- numeric(n)
  for (i in 1:n) {
    soma <- if (i > 1) sum(LU[i, 1:(i - 1)] * y[1:(i - 1)]) else 0
    y[i] <- b[i] - soma}

  # Retrosubstituição
  s <- numeric(n)
  for (i in n:1) {
    soma <- if (i < n) sum(LU[i, (i + 1):n] * s[(i + 1):n]) else 0
    s[i] <- (y[i] - soma) / LU[i, i]}
  return(s)}
```

Método de Newton

```
newton_system <- function(F_func, J_func, x0, tol = 1e-4, max_iter = 50) {
  x_k <- x0

  for (k in 1:max_iter) {
    F_xk <- F_func(x_k)

    if (max(abs(F_xk)) < tol) {
      cat(sprintf("Convergência atingida na iteração %d", k))
      return(list(solucao = x_k, iteracoes = k))}

    J_xk <- J_func(x_k)
    s_k <- solve_gauss_pivot(J_xk, -F_xk)
    x_k_plus_1 <- x_k + s_k

    if (max(abs(s_k)) < tol) {
      cat(sprintf("Convergência atingida na iteração %d", k))
      return(list(solucao = x_k_plus_1, iteracoes = k))}
    x_k <- x_k_plus_1}
```

```
  warning(sprintf("Método não convergiu em %d iterações.\n", max_iter))
  return(list(solucao = x_k, iteracoes = max_iter))
}
```

Método de Newton Modificado

```
newton_modificado_lu <- function(F_func, J_func, x0, tol = 1e-4, max_iter = 50) {
  x_k <- x0
  J_x0 <- J_func(x0)
  LU_J0 <- lu_decompose(J_x0)

  for (k in 1:max_iter) {
    F_xk <- F_func(x_k)

    if (max(abs(F_xk)) < tol) {
      cat(sprintf("Convergência atingida na iteração %d", k))
      return(list(solucao = x_k, iteracoes = k))}

    s_k <- lu_solve(LU_J0, -F_xk)
    x_k_plus_1 <- x_k + s_k

    if (max(abs(s_k)) < tol) {
      cat(sprintf("Convergência atingida na iteração %d", k))
      return(list(solucao = x_k_plus_1, iteracoes = k))}
    x_k <- x_k_plus_1}

  warning(sprintf("Método não convergiu em %d iterações.\n", max_iter))
  return(list(solucao = x_k, iteracoes = max_iter))
}
```

2. Resolva os sistemas não lineares abaixo usando seus programas para os métodos de Newton e Newton Modificado com $\epsilon = 10^{-4}$

d) (Função de Rosenbrock) [24]

$$\begin{cases} 10(x_2 - x_1^2) = 0 \\ 1 - x_1 = 0 \end{cases} \qquad x^{(0)=(-1.2,1)^\top}$$

```
F_rosenbrock <- function(x) {
  f1 <- 10 * (x[2] - x[1]^2)
  f2 <- 1 - x[1]
  return(c(f1, f2))}

J_rosenbrock <- function(x) {
  jacobiana <- matrix(c(-20 * x[1], -1, 10, 0), nrow = 2, ncol = 2)
  return(jacobiana)}

x_inicial <- c(-1.2, 1)
tolerancia <- 1e-4
```

```r
resultado_newton <- newton_system(F_rosenbrock, J_rosenbrock, x_inicial, tol = tolerancia)
```

```
## Convergência atingida na iteração 3
```

```r
resultado_newton_mod <- newton_modificado_lu(F_rosenbrock, J_rosenbrock, x_inicial,
                                             tol = tolerancia)
```

```
## Convergência atingida na iteração 3
```

```r
cat("\nMétodo de Newton:\n")
```

```
##
## Método de Newton:
```

```r
print(resultado_newton)
```

```
## $solucao
## [1] 1 1
##
## $iteracoes
## [1] 3
```

```r
cat("\nMétodo de Newton Modificado:\n")
```

```
##
## Método de Newton Modificado:
```

```r
print(resultado_newton_mod)
```

```
## $solucao
## [1] 1 1
##
## $iteracoes
## [1] 3
```

e) (Broyden Tridiagonal) [24]

$$\begin{cases} f_1(x) = (3-2x_1)x_1 - 2x_2 + 1 = 0 \\ f_i(x) = (3-2x_i)x_i - x_{i-1} - 2x_{i+1} + 1 = 0, \quad i = 2, \dots, 9 \\ f_{10}(x) = (3-2x_{10})x_{10} - x_9 + 1 = 0 \end{cases} \qquad x^{(0)} = (-1, -1, \dots, -1)^\top$$

```r
F_broyden <- function(x) {
  n <- length(x)
  f <- numeric(n)

  f[1] <- (3 - 2 * x[1]) * x[1] - 2 * x[2] + 1
  for (i in 2:(n - 1)) {
    f[i] <- (3 - 2 * x[i]) * x[i] - x[i - 1] - 2 * x[i + 1] + 1}
```

```
  f[n] <- (3 - 2 * x[n]) * x[n] - x[n - 1] + 1

  return(f)}

J_broyden <- function(x) {
  n <- length(x)
  J <- matrix(0, nrow = n, ncol = n)

  for (i in 1:n) {
    # Diagonal principal
    J[i, i] <- 3 - 4 * x[i]
    # Diagonal inferior
    if (i > 1) {
      J[i, i - 1] <- -1}
    # Diagonal superior
    if (i < n) {
      J[i, i + 1] <- -2}}
  return(J)}

x_inicial_broyden <- rep(-1, 10)
tolerancia <- 1e-4

resultado_newton_e <- newton_system(F_broyden, J_broyden, x_inicial_broyden,
                                    tol = tolerancia)
```

```
## Convergência atingida na iteração 4
```

```
resultado_newton_mod_e <- newton_modificado_lu(F_broyden, J_broyden, x_inicial_broyden,
                                               tol = tolerancia)
```

```
## Convergência atingida na iteração 9
```

```
cat("\nMétodo de Newton:\n")
```

```
##
## Método de Newton:
```

```
print(resultado_newton_e)
```

```
## $solucao
##  [1] -0.5707249 -0.6818091 -0.7022118 -0.7055124 -0.7049086 -0.7015005
##  [7] -0.6918964 -0.6658091 -0.5960545 -0.4164305
##
## $iteracoes
## [1] 4
```

```
cat("\nMétodo de Newton Modificado:\n")
```

```
##
## Método de Newton Modificado:
```

```r
print(resultado_newton_mod_e)
```

```
## $solucao
##  [1] -0.5707280 -0.6818116 -0.7022144 -0.7055157 -0.7049134 -0.7015078
##  [7] -0.6919073 -0.6658250 -0.5960765 -0.4164563
##
## $iteracoes
## [1] 9
```