Métodos Computacionais para Estatística e Otimização

Lista 6 - Luiz Henrique Barretta Francisco

1. Considere o método regula-falsi para a solução de uma equação não linear. Forneça uma implementação em R deste método e use sua implementação para resolver a seguinte equação não linear.

$$D(\theta) = 2n[\log(\frac{\hat{\theta}}{\theta}) + \bar{y}(\theta - \hat{\theta})] \leq 3.84$$

```
regula_falsi <- function(f, a, b, tol = 1e-6, max_iter = 1000) {
  fa <- f(a)
  fb <- f(b)
  if (fa * fb > 0) {
    stop("Os valores de f(a) e f(b) devem ter sinais opostos.")}

  for (i in 1:max_iter) {
    c <- (a * fb - b * fa) / (fb - fa)
    fc <- f(c)
    if (abs(fc) < tol || abs(b - a) < tol) {
      return(c)}
    if (fa * fc < 0) {
      b <- c
      fb <- fc
    } else {
      a <- c
      fa <- fc}
  }
  stop("Número máximo de iterações alcançado sem convergência.")
}

D <- function(theta, n, theta_hat, y_bar) {
  2 * n * (log(theta_hat / theta) + y_bar * (theta - theta_hat)) - 3.84}

n <- 100
theta_hat <- 2
y_bar <- 1.5

D_theta <- function(theta) {
  D(theta, n, theta_hat, y_bar)}

raiz <- regula_falsi(D_theta, a = 0.5, b = 5)
raiz
```

```
## [1] 2.019154
```

2. Use o método de Newton para resolver o seguinte sistema de equações não lineares

$$y = cos(x)$$
$$x = sen(y)$$

Forneça a implementação do método de Newton e a aplicação ao sistema dado. Use para comparação as rotinas prontas do pacote *rootSolve*.

```
library(rootSolve)
library(bench)
newton_sistema <- function(F, J, x0, tol = 1e-6, max_iter = 100) {
  x <- x0
  for (i in 1:max_iter) {
    F_val <- F(x)
    if (sqrt(sum(F_val^2)) < tol) {
      return(list(raiz = x, iteracoes = i, convergiu = TRUE))
    }
    J_val <- J(x)
    delta <- try(solve(J_val, -F_val), silent = TRUE)
    if (inherits(delta, "try-error")) {
      stop("Jacobiano singular na iteração ", i)
    }
    x <- x + delta
    if (sqrt(sum(delta^2)) < tol * (sqrt(sum(x^2)) + 1e-12)) {
      return(list(raiz = x, iteracoes = i, convergiu = TRUE))}
  }
  warning("Máximo de iterações alcançado sem convergência.")
  return(list(raiz = x, iteracoes = max_iter, convergiu = FALSE))}

sistema <- function(v) {
  x <- v[1]
  y <- v[2]
  c(y - cos(x),
    x - sin(y))}

jacobiano <- function(v) {
  x <- v[1]
  y <- v[2]
  matrix(c(sin(x), 1,
           1, -cos(y)),
    nrow = 2, byrow = TRUE)}

x0 <- c(0.5, 0.5)

resultado_newton <- newton_sistema(sistema, jacobiano, x0)
cat("Solução pelo método de Newton:", round(resultado_newton$raiz, 6), "\n")
```

```
## Solução pelo método de Newton: 0.69482 0.768169
```

```
resultado_rootSolve <- multiroot(
  f = sistema,
  start = x0,
  jacfunc = jacobiano,
```

```
  ctol = 1e-12)
cat("Solução pelo rootSolve:", round(resultado_rootSolve$root, 6), "\n")
```

```
## Solução pelo rootSolve: 0.69482 0.768169
```

```
resultado_bench <- bench::mark(
  Método_Customizado = {
    newton_sistema(sistema, jacobiano, x0, tol = 1e-12, max_iter = 100)},
  rootSolve = {
    multiroot(f = sistema, start = x0, jacfunc = jacobiano, ctol = 1e-12)},
  check = FALSE, iterations = 100)
print(resultado_bench)
```

```
## # A tibble: 2 x 13
##   expression      min median `itr/sec` mem_alloc `gc/sec` n_itr  n_gc total_time
##   <bch:expr>   <bch:> <bch:>     <dbl> <bch:byt>    <dbl> <int> <dbl>   <bch:tm>
## 1 Método_Cust~ 97.7us  114us     2681.        0B     27.1    99     1     36.9ms
## 2 rootSolve    86.9us  106us     7763.      856B      0     100     0     12.9ms
## # i 4 more variables: result <list>, memory <list>, time <list>, gc <list>
```

3. Implemente o método quasi-Newton BFGS. Use a sua implementação para otimizar a seguinte função perda:

$$L(y, \mu) = \sum_{i=1}^{n} \log\left(cosh(\mu_i - y_i)\right)$$

Considere o seguinte código para gerar y_i. O parâmetro deve ser especificado como uma reta $\mu_i = \beta_0 + \beta_1 x_i$. Note que você deverá otimizar os parâmetros β_0 e β_1 que na simulação foram fixados em $\beta_0 = 5$ e $\beta_1 = 3$.

```
loss <- function(beta, x, y) {
  mu <- beta[1] + beta[2] * x
  sum(log(cosh(mu - y)))}

grad <- function(beta, x, y) {
  mu <- beta[1] + beta[2] * x
  residuals <- mu - y
  d_beta0 <- sum(tanh(residuals))
  d_beta1 <- sum(tanh(residuals) * x)
  c(d_beta0, d_beta1)}

BFGS <- function(theta, x, y, max_iter = 1000, tol = 1e-6) {
  n_params <- length(theta)
  H <- diag(n_params)
  grad_old <- grad(theta, x, y)

  for (i in 1:max_iter) {
    if (sqrt(sum(grad_old^2)) < tol) break
    d <- as.vector(-H %*% grad_old)
    alpha <- 1
    for (ls_iter in 1:50) {
      theta_new <- theta + alpha * d
      loss_new <- loss(theta_new, x, y)
```

```
      loss_old <- loss(theta, x, y)
      if (loss_new <= loss_old + 0.0001 * alpha * sum(grad_old * d)) break
      alpha <- alpha * 0.5}
    theta_new <- theta + alpha * d
    grad_new <- grad(theta_new, x, y)
    s <- theta_new - theta
    y_grad <- grad_new - grad_old
    sy_dot <- sum(y_grad * s)

    if (sy_dot > .Machine$double.eps) {
      rho <- 1 / sy_dot
      H <- (diag(n_params) - rho * outer(s, y_grad)) %*% H %*%
           (diag(n_params) - rho * outer(y_grad, s)) + rho * outer(s, s)}
    theta <- theta_new
    grad_old <- grad_new}
  list(theta = theta, iteracoes = i)}

set.seed(123)
x1 <- rnorm(100)
mu <- 5 + 3 * x1
y <- rt(n = 100, df = 3) + mu

resultado <- BFGS(theta = c(0, 0), x = x1, y = y)

optim_result <- optim(par = c(0, 0), fn = loss, gr = grad,
                      x = x1, y = y, method = "BFGS")

cat("Implementação BFGS: ", round(resultado$theta, 4))
```

```
## Implementação BFGS:  4.983 2.7715
```

```
cat("Resultado do optim(): ", round(optim_result$par, 4))
```

```
## Resultado do optim():  4.983 2.7715
```

4. Considere o conjunto de dados youtube.csv [http://leg.ufpr.br/~wagner/data/] que apresenta o número de views e inscritos de dois canais de sucesso do youtube desde o dia de sua abertura. O objetivo é predizer o número acumulado de inscritos em cada um destes canais para o próximo ano (365 dias). Para isto você decidiu emprestar um modelo biológico que modela o crescimento de bactérias chamado de modelo logístico, dado pela seguinte equação:

$$y = \frac{L}{1 + \exp\left(\beta(-\beta_0)\right)}$$

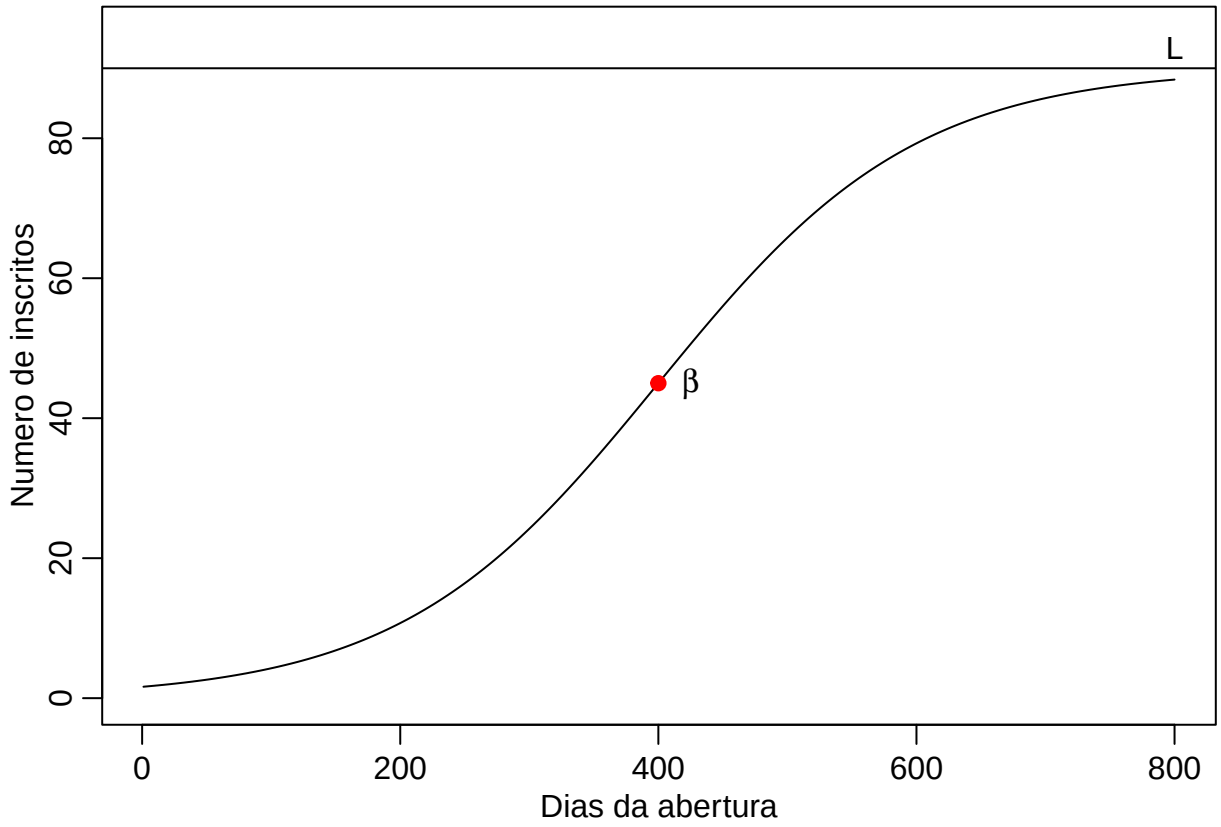
onde L é o valor máximo da curva (platô), β_0 é o valor de x no ponto médio da curva (tempo de meia-vida) e β é a declividade da curva. A Figura abaixo apresenta um gráfico do modelo logístico.

```
par(mfrow = c(1,1), mar=c(2.6, 3, 1.2, 0.5), mgp = c(1.6, 0.6, 0))

f_log <- function(DIAS, L, beta, beta0) {
  out <- L/(1+ exp(-beta*(DIAS - beta0)))
```

```
  return(out)}

DIAS <- 1:800
plot(f_log(DIAS = DIAS, L = 90, beta = 0.01, beta0 = 400) ~ DIAS,
     ylab = "Numero de inscritos", xlab = "Dias da abertura",
     type = "l", ylim = c(0,95))
abline(h = 90)
text(x = 800, y = 93, label = "L")
text(x = 425, y = f_log(DIAS = 400, L = 90, beta = 0.01, beta0 = 400),
     label = expression(beta))
points(x = 400, pch = 19, col = "red",
       y = f_log(DIAS = 400, L = 90, beta = 0.01, beta0 = 400))
```



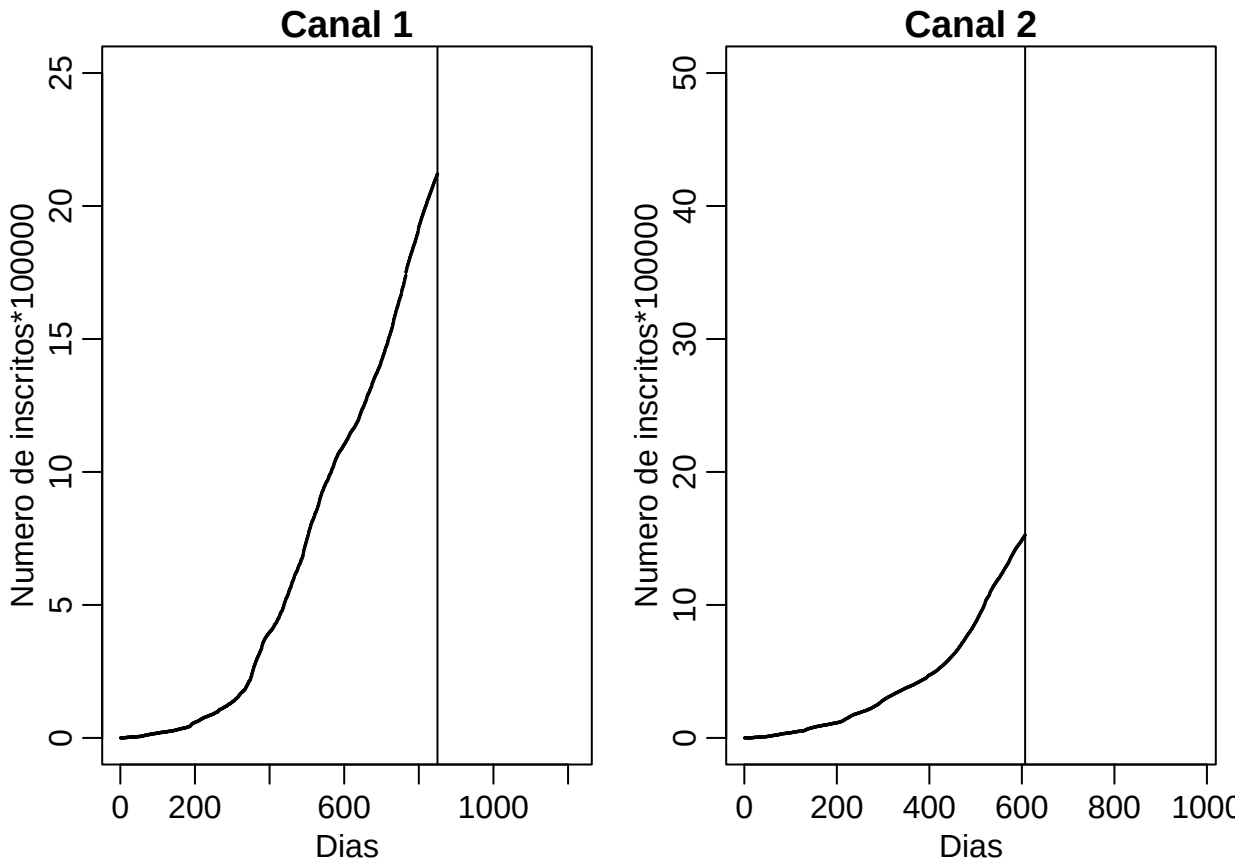
Note que o modelo representa a intuição de como o número acumulado de inscritos em um canal deve se comportar. No Código abaixo a base de dados é carregada e organizada por canal.

```
url <- "http://leg.ufpr.br/~wagner/data/youtube.txt"
dados <- read.table(url, header = TRUE)
dados_canal <- split(dados, dados$CANAL)
dados1 <- dados_canal[[1]]
dados2 <- dados_canal[[2]]
dados1$INSCRITOS <- dados1$INSCRITOS/100000
dados1$Y <- cumsum(dados1$INSCRITOS)
dados2$INSCRITOS <- dados2$INSCRITOS/100000
dados2$Y <- cumsum(dados2$INSCRITOS)
```

Podemos fazer o gráfico dos dados observados para explicitar o objetivo de predizer o número de inscritos acumulado para os próximos 365 dias.

```
par(mfrow = c(1,2), mar=c(2.6, 3, 1.2, 0.5), mgp = c(1.6, 0.6, 0))
plot(dados1$Y ~ dados1$DIAS, xlim = c(0, 1215), ylim = c(0, 25),
     ylab = "Numero de inscritos*100000", main = "Canal 1",
     xlab = "Dias", type = "o", cex = 0.1)
abline(v = 850)

plot(dados2$Y ~ dados2$DIAS, ylab = "Numero de inscritos*100000", main = "Canal 2",
     xlab = "Dias", ylim = c(0, 50), xlim = c(0, 980), type = "p", cex = 0.1)
abline(v = 607)
```



Escolha um destes canais e otimize o modelo logístico para predizer qual será o número acumulado de inscritos para os próximos 365 dias. Ao apresentar sua solução computacional faça o máximo para explicar as suas decisões e estratégias de implementação. Tome o máximo de cuidado para que a sua análise seja reproduzível.

```
modelo_logistico <- function(DIAS, L, beta, beta0) {
  L / (1 + exp(-beta * (DIAS - beta0)))}

set.seed(123)
ajuste <- nls(Y ~ modelo_logistico(DIAS, L, beta, beta0),
              data = dados2,
              start = list(L = 25, beta = 0.01, beta0 = 400),
              algorithm = "port",
```

```
            lower = list(L = 20, beta = 0.001, beta0 = 300),
            upper = list(L = 30, beta = 0.1, beta0 = 500))
summary(ajuste)
```

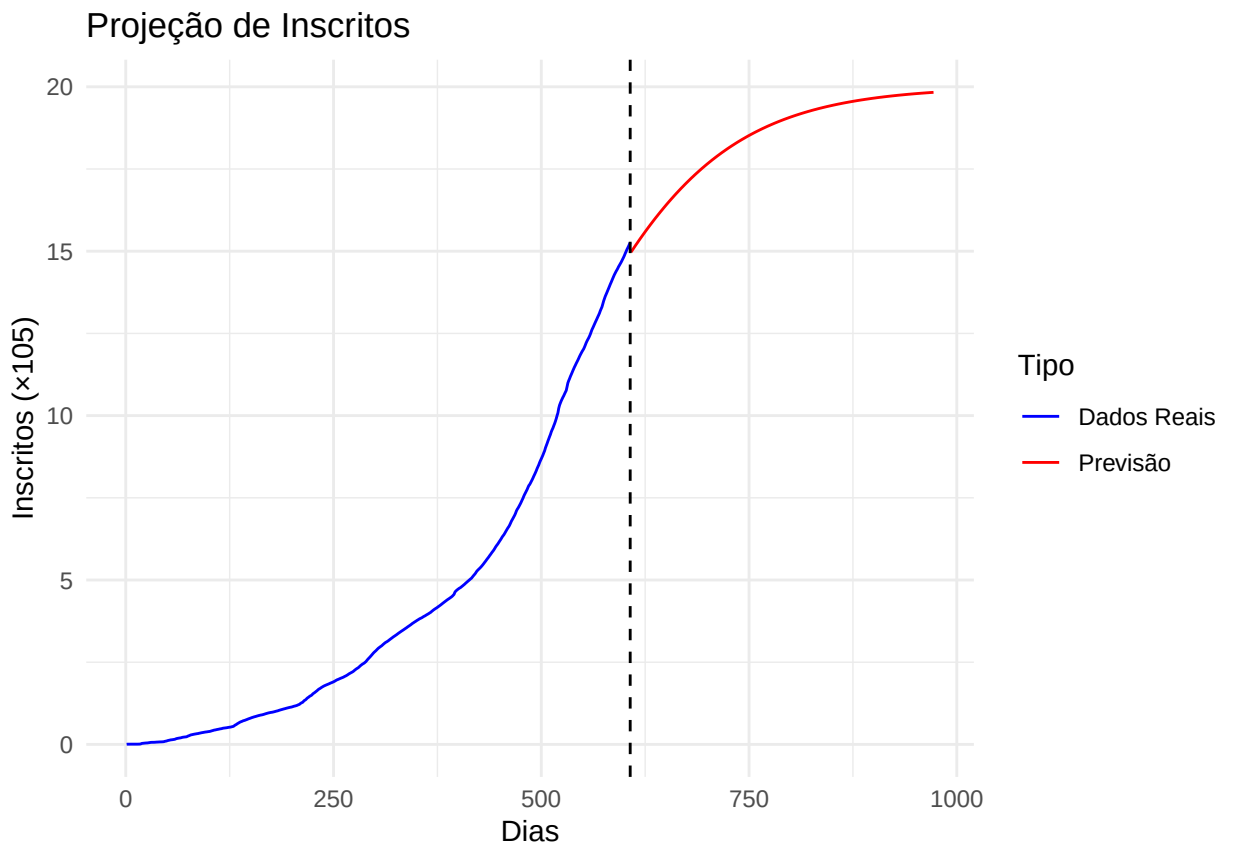
```
##
## Formula: Y ~ modelo_logistico(DIAS, L, beta, beta0)
##
## Parameters:
##        Estimate Std. Error t value Pr(>|t|)
## L     2.000e+01  5.159e-01   38.77   <2e-16 ***
## beta  1.010e-02  2.098e-04   48.15   <2e-16 ***
## beta0 5.000e+02  5.756e+00   86.87   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6138 on 604 degrees of freedom
##
## Algorithm "port", convergence message: both X-convergence and relative convergence (5)
```

```
ultimo_dia <- max(dados2$DIAS)
dias_futuro <- data.frame(DIAS = (ultimo_dia + 1):(ultimo_dia + 365))
previsao <- predict(ajuste, newdata = dias_futuro)

library(ggplot2)

dados_completos <- data.frame(
  DIAS = c(dados2$DIAS, dias_futuro$DIAS),
  Y = c(dados2$Y, previsao),
  Tipo = c(rep("Dados Reais", length(dados2$DIAS)), rep("Previsão", length(dias_futuro$DIAS))))

ggplot(dados_completos, aes(x = DIAS, y = Y, color = Tipo)) +
  geom_line() + geom_vline(xintercept = ultimo_dia, linetype = "dashed") +
  scale_color_manual(values = c("blue", "red")) +
  labs(title = "Projeção de Inscritos", x = "Dias", y = "Inscritos (×10 )") +
  theme_minimal()
```




```r
library(nls2)
grid_par <- expand.grid(
  L = seq(24, 26, length = 10),
  beta = seq(0.005, 0.02, length = 10),
  beta0 = seq(380, 420, length = 10))

ajuste_grid <- nls2(Y ~ modelo_logistico(DIAS, L, beta, beta0),
                    data = dados2, start = grid_par, algorithm = "brute-force")

perda <- function(theta) {
  L <- theta[1]
  beta <- theta[2]
  beta0 <- theta[3]
  y_pred <- modelo_logistico(dados2$DIAS, L, beta, beta0)
  sum((dados2$Y - y_pred)^2)}

optim_result <- optim(c(25, 0.01, 400), perda, method = "BFGS")
```

O código apresentado tem como objetivo otimizar um modelo logístico para prever o número acumulado de inscritos nos próximos 365 dias. O modelo logístico é ajustado usando a função *nls()* (non-linear least squares), onde os parâmetros L, β, e β_0 são estimados a partir dos dados disponíveis (armazenados em dados2). As estimativas dos parâmetros indicam que o valor de L é significativamente determinado como 20, β como 0.01, e β_0 como 500, com todos os parâmetros sendo altamente significativos (valores de $p < 0.001$). Esse ajuste foi realizado com a metodologia de "port" do *nls()*, que limita os parâmetros dentro de intervalos pré-definidos para evitar sobreajustes.

Além disso, o modelo foi validado por meio de uma previsão para os próximos 365 dias utilizando a função *predict()*. O gráfico gerado com *ggplot2* visualiza tanto os dados reais quanto as previsões, destacando a linha vertical no último dia de dados conhecidos. Para otimizar ainda mais o modelo, a busca por melhores parâmetros foi realizada com uma abordagem de "brute-force" usando a função *nls2* e, posteriormente, a função *optim()* foi aplicada para ajustar os parâmetros de forma a minimizar o erro quadrático. Esse processo de otimização é eficaz, pois busca as melhores estimativas para os parâmetros, minimizando a diferença entre os valores previstos e os reais.