Inferência Estatística para Ciência de Dados

# Solução numérica de sistemas de equações

*Exemplos: Função perda absoluta*

*Wagner Hugo Bonat*

*2018-04-06*

# 1 Redução de dados usando perda absoluta

## 1.1 Mediana via perda absoluta

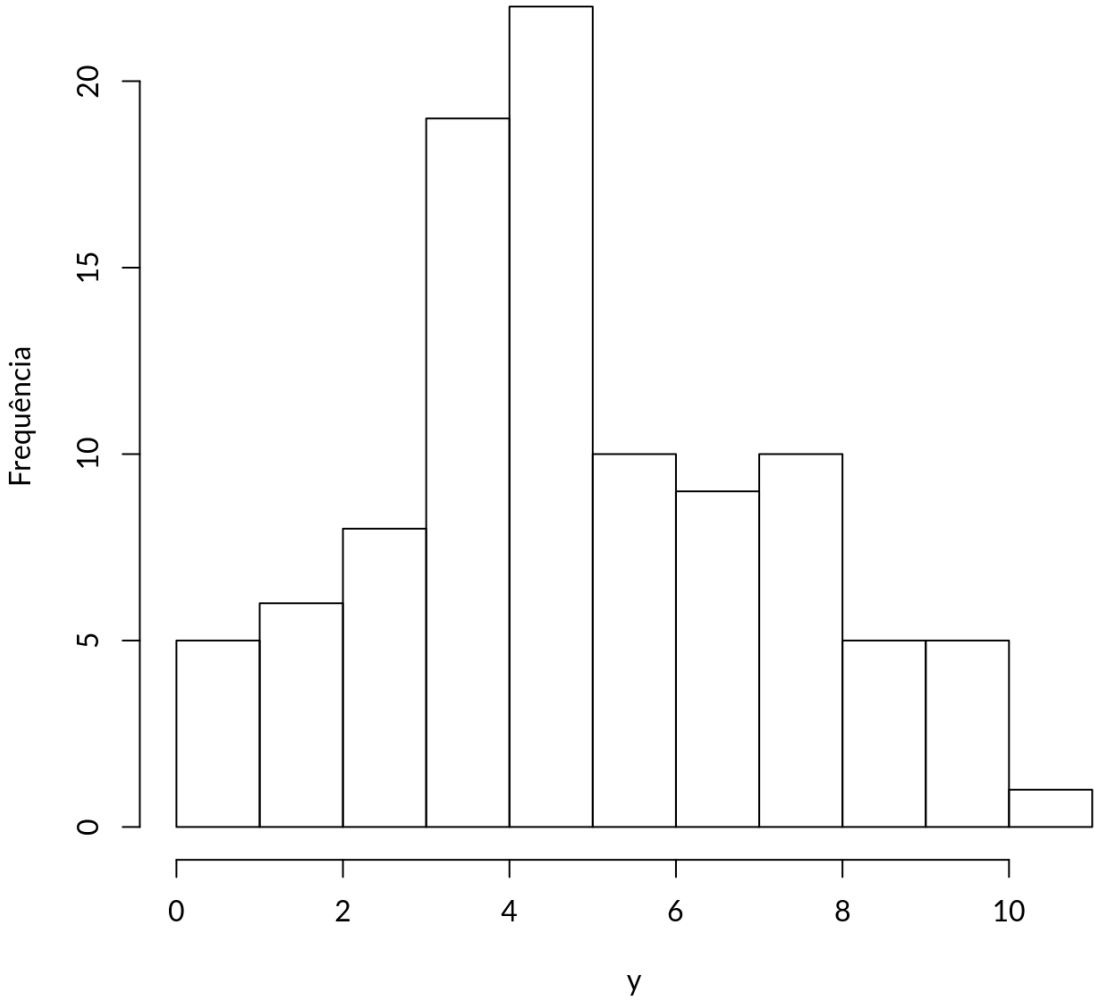
Seja $y_i$ um conjunto de observações de uma variável de interesse. Suponha que o interesse seja resumir a informação contida em $y_i$ através de um único número denotado por $\mu$. Existem várias alternativas para isso, neste exemplo vamos adotar a função perda absoluta. Assim, o problema se resume a encontrar o valor digamos $\hat{\mu}$, tal que a função perda absoluta,

$$f(\mu) = \sum_{i=1}^{n} |y_i - \mu|,$$

seja o menor possível.

A abordagem mais simples para este caso é traçar o gráfico da função e baseado neste gráfico encontrar o valor de $\mu$ que torna $f(\mu)$ mínima. Para começar, vamos supor que os dados que temos para resumir são os seguintes.

```
y <- rpois(100, lambda = 5)
hist(y,  main = "", ylab = "Frequência")
```




O próximo passo é implementar em a função perda absoluta.

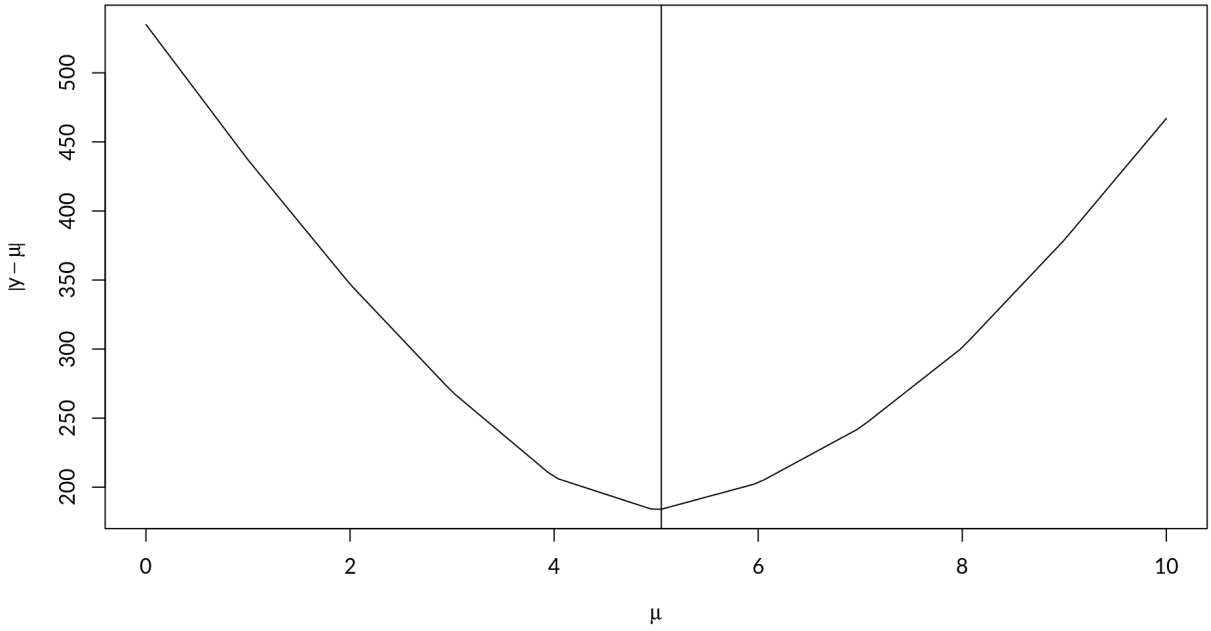
```
perda_abs <- function(mu, y) sum( abs(y - mu))
```

Podemos facilmente avaliar a função perda absoluta para um gride de valores de $\mu$ e verificar qual deles à torna mínima.

```
gride_mu <- seq(0, 10, l = 100)
perda <- c()
for(i in 1:100) {
  perda[i] <- perda_abs(mu = gride_mu[i], y = y)
}
temp <- data.frame(gride_mu, perda)
head(temp)
#    gride_mu    perda
# 1 0.0000000 535.0000
# 2 0.1010101 525.1010
# 3 0.2020202 515.2020
# 4 0.3030303 505.3030
# 5 0.4040404 495.4040
# 6 0.5050505 485.5051
temp[which.min(temp$perda),]
#    gride_mu    perda
# 51 5.050505 184.0101
```

Podemos, representar isso graficamente.

```
plot(perda ~ gride_mu, type = "l", ylab = expression(abs(y - mu)),
     xlab = expression(mu))
abline(v = temp[which.min(temp$perda),]$gride_mu)
```




O método gráfico é extremamente simples e intuitivo, porém não é generalizável para funções perda com quatro ou mais parâmetros.

A forma usual de proceder é usar Cálculo diferencial integral para achar o ponto de inflexão de $f(\mu)$. Nesta abordagem, o primeiro passo é obter a derivada de $f(\mu)$ em relação a $\mu$, ou seja,

$$f'(\mu) = \sum_{i=1}^{n} \frac{y_i - \mu}{|y_i - \mu|}(-1).$$

Note que $f'(\mu)$ apresenta apenas dois valores $-1$ e $1$, sendo assim, precisamos encontrar o valor de $\hat{\mu}$ que faça com que exatamente $50\%$ dos valores de $y_i$ sejam menores que $\hat{\mu}$ e exatamente $50\%$ dos valores de $y_i$ sejam maiores que $\hat{\mu}$. Esse valor é a conhecida mediana dos dados. Assim, o valor que

torna $f(\mu)$ mínima é a mediana. Note entretanto, que não é possível obter a mediana por resolver $f'(\mu) = 0$. Sendo assim, podemos resolver essa equação numéricamente usando um dos métodos vistos em sala.

Como temos um problema unidimensional os métodos baseados na ideia de confinamento são bastante simples e seguros. Foram apresentados os métodos da Bisseção e Regula Falsi, cujas implementações em são apresentadas abaixo.

```
# Método da Bisseção

bissecao <- function(fx, a, b, tol = 1e-04, max_iter = 100, ...) {
  fa <- fx(a, ...);  fb <- fx(b, ...)
  if(fa*fb > 0) stop("Solução não está no intervalo")
  solucao <- c()
  sol <- (a + b)/2
  solucao[1] <- sol
  limites <- matrix(NA, ncol = 2, nrow = max_iter)
  for(i in 1:max_iter) {
    test <- fx(a, ...)*fx(sol, ...)
    if(test < 0) {
      solucao[i+1] <- (a + sol)/2
      b = sol
      }
    if(test > 0) {
      solucao[i+1] <- (b + sol)/2
      a = sol
      }
    if( abs( (b-a)/2) < tol) break
    sol = solucao[i+1]
    limites[i,] <- c(a,b)
  }
  out <- list("Tentativas" = solucao, "Limites" = limites, "Raiz" = solucao[i+1])
  return(out)
}

# Método Regula Falsi

regula_falsi <- function(fx, a, b, tol = 1e-04, max_iter = 100, ...) {
  fa <- fx(a, ...);  fb <- fx(b, ...)
  if(fa*fb > 0) stop("Solução não está no intervalo")
  solucao <- c()
  sol <- (a*fx(b, ...) - b*fx(a, ...))/(fx(b, ...) - fx(a, ...))
  solucao[1] <- sol
  limites <- matrix(NA, ncol = 2, nrow = max_iter)
  for(i in 1:max_iter) {
    test <- fx(a, ...)*fx(sol, ...)
    if(test < 0) {
      b = sol
      solucao[i+1] <- (a*fx(b,...) - b*fx(a, ...))/(fx(b,...) - fx(a, ...))
      }
    if(test > 0) {
      a = sol
      solucao[i+1] <- sol <- (a*fx(b, ...) - b*fx(a, ...))/(fx(b,...) - fx(a,
 ...))
      }
    if( abs(solucao[i+1] - solucao[i]) < tol) break
    sol = solucao[i+1]
    limites[i,] <- c(a,b)
  }
  out <- list("Tentativas" = solucao, "Limites" = limites, "Raiz" = sol)
  return(out)
}
```

Podemos, usar as funções acima para encontrar a raiz $\hat{\mu}$ que satisfaz a equação $f'(\hat{\mu}) = 0$. Para isto, primeiro implementamos $f'(\mu)$ e depois aplicamos os métodos para encontrar a raiz $\hat{\mu}$.

```
fx <- function(mu, y) -sum(sign(y-mu))
# Bisseção
sol_bis <- bissecao(fx = fx, a = 1, b = 9, y = y)
sol_bis$Raiz
# [1] 5.000061

# Regula Falsi
sol_rf <- regula_falsi(fx = fx, a = 1, b = 9, y = y)
sol_rf$Raiz
# [1] 4.999997
```

Para ter ainda mais intuição sobre como os métodos numéricos atuam, vamos traçar o caminho que o algoritmo percorreu para encontrar a solução ótima.

```
plot(perda ~ gride_mu, type = "l", ylab = expression(abs(y - mu)),
     xlab = expression(mu))
#, ylim = c(150, 200), xlim = c(4, 6))
segments(x0 = 1, y0 = perda_abs(1, y = y), x1 = 9, y1 = perda_abs(9, y = y))
for(i in 1:15) {
  pt1 <- sol_bis$Limites[i,1]
  pt2 <- sol_bis$Limites[i,2]
  segments(x0 = pt1 , y0 = perda_abs(pt1, y = y), x1 = pt2,
           y1 = perda_abs(pt2, y = y), col = i+1)
  #Sys.sleep(1)
}
```

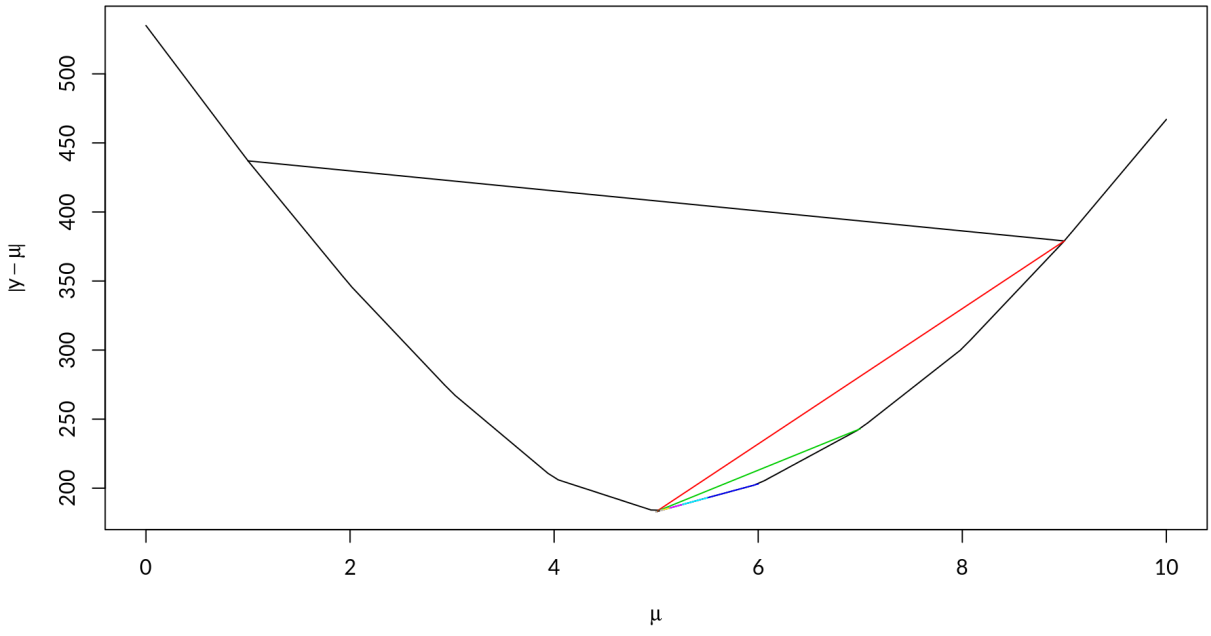



Gráfico similar pode ser feito para o traçado do método regula falsi.

# 1.2 Regressão linear simples via perda absoluta

No exemplo anterior nós reduzimos ou resumimos nosso conjunto de dados através de um único número. Quando uma outra variável está presente podemos relacionar os valores da variável de principal interesse com o da covariável, digamos $x_i$ através de uma reta, ou seja,

$$y_i = \beta_0 + \beta_1 x_i.$$

Note que agora ao invés de um resumo de um único número o resumo dos dados é dado por um reta. Nosso objetivo agora é encontrar qual reta melhor descreve nosso conjunto de observações. Novamente, precisamos de uma regra para escolher qual é a "melhor reta" para o conjunto de observações em análise.

Neste exemplo, vamos adotar a perda absoluta que consiste em encontrar os valores de $\beta_0$ e $\beta_1$ que minimizam a função perda absoluta, ou seja,

$$f(\beta_0, \beta_1) = \sum_{i=1}^{n} |y_i - (\beta_0 + \beta_1 x_i)|.$$

O primeiro passo é obter o vetor gradiente de $f(\beta_0, \beta_1)$. Neste caso, temos

$$\nabla f(\beta_0, \beta_1) = \left( \frac{\partial f(\beta_0, \beta_1)}{\partial \beta_0}, \frac{\partial f(\beta_0, \beta_1)}{\partial \beta_1} \right)^\top,$$

onde

$$\frac{\partial f(\beta_0, \beta_1)}{\partial \beta_0} = \sum_{i=1}^{n} \frac{y_i - \beta_0 - \beta_1 x_i}{|y_i - \beta_0 - \beta_1 x_i|}(-1) \quad \text{e} \quad \frac{\partial f(\beta_0, \beta_1)}{\partial \beta_0} = \sum_{i=1}^{n} \frac{y_i - \beta_0 - \beta_1 x_i}{|y_i - \beta_0 - \beta_1 x_i|}(-x_i).$$

Assim, precisamos encontrar $\beta_0$ e $\beta_1$ que satisfaçam o seguinte sistema

$$\begin{aligned} \textstyle\sum_{i=1}^{n} \frac{y_i - \beta_0 - \beta_1 x_i}{|y_i - \beta_0 - \beta_1 x_i|}(-1) &= 0 \\ \textstyle\sum_{i=1}^{n} \frac{y_i - \beta_0 - \beta_1 x_i}{|y_i - \beta_0 - \beta_1 x_i|}(-xi) &= 0. \end{aligned}$$

Agora temos um sistema com duas equações e duas incógnitas. Assim, os métodos de Newton e Gradiente descendente são adequados. Para usar o método de Newton precisamos obter a matriz Jacobiana,

$$\mathbf{J} = \begin{bmatrix} \frac{\partial^2 f(\beta_0,\beta_1)}{\partial \beta_0^2} & \frac{\partial^2 f(\beta_0,\beta_1)}{\partial \beta_0 \beta_1} \\ \frac{\partial^2 f(\beta_0,\beta_1)}{\partial \beta_1 \beta_0} & \frac{\partial^2 f(\beta_0,\beta_1)}{\partial \beta_1^2} \end{bmatrix}.$$

Entretanto, como $f'(\mu)$ só depende do sinal de $y_i - \beta_0 - \beta_1 x_i$, todas as entradas de $\mathbf{J}$ são $0$ e portanto, não informa nada sobre o comportamento de $f(\mu)$. Isso também mostra que o método de Newton não se aplica neste caso. Porém, o método do gradiente descendente não precisa da matriz Jacobiana o que o torna um bom candidato para resolver o sistema em questão. O código abaixo implementa o método gradiente descendente para uma função genérica.

```
grad_des <- function(fx, x1, alpha, max_iter = 100, tol = 1e-04, ...) {
  solucao <- matrix(NA, ncol = length(x1), nrow = max_iter)
  solucao[1,] <- x1
  for(i in 1:c(max_iter-1)) {
    solucao[i+1,] <- solucao[i,] + alpha*fx(solucao[i,], ...)
    #print(solucao[i+1,])
    if( sum(abs(solucao[i+1,] - solucao[i,])) < tol) break
  }
  return(list("It" = na.exclude(solucao), "Sol" = solucao[i+1,]))
}
```

Vamos criar um conjunto de dados simulados para ilustrar o uso do método.

```
# Covariável
set.seed(123)
x <- seq(0, 1, l = 100)
beta_vec <- c(2, 5)
mu <- 2 + 5*x
y <- rnorm(y, mean = mu, sd = 1)
```

Agora implementamos o sistema de equações a ser resolvido.

```
my_grad <- function(beta_vec, y = y, x = x) {
  mu <- beta_vec[1] + beta_vec[2]*x
  sinais <- sign(y - mu)
  d_b0 <- sum(sinais*-1)
  d_b1 <- sum(sinais*(-x))
  return(c(d_b0, d_b1))
}
```

Por fim, usamos nossa implementação do método gradiente descendente para resolver o sistema.

```
tt = grad_des(fx = my_grad, alpha = -0.01, x1 = c(0,0),
              max_iter = 1000, tol = 1e-02, y = y, x = x)
# Conferindo
my_grad(beta_vec = tt$Sol, y = y, x = x)
# [1]  0.0000000 -0.2626263
```

Note que a solução encontrada resolve o sistema apenas aproximadamente. Isso é devido ao comportamento plano da função próximo ao mínimo fazendo com que tenha várias soluções aproximadas. A Figura abaixo ilustra o traço do algoritmo até encontrar o mínimo da função objetivo.

```
perda_abs_reg <- function(beta_vec, y, x1) {
  mu <- beta_vec[1] + beta_vec[2]*x1
  out <- sum(abs(y - mu))
  return(out)
}
beta0 <- seq(-1,5, l = 100)
beta1 <- seq(0,10, l = 100)
grid_beta <- expand.grid(beta0, beta1)
pp <- matrix(apply(grid_beta, 1, perda_abs_reg, y = y, x1 = x), 100, 100)
image(x = beta0, y = beta1, z = pp, col = heat.colors(15),
      ylab = expression(beta[1]), xlab = expression(beta[0]),
      breaks = c(seq(72,150, 8), 200, 225, 250, 300, 350, 400) )
contour(x = beta0, y = beta1, z = pp, add = TRUE,
        breaks = c(seq(72,150, 8), 200, 225, 250, 300, 350, 400))
points(x = tt$Sol[1], y = tt$Sol[2], pch = 19)
for(i in 1:62) {
  text(x = tt$It[i,1], y = tt$It[i,2], label = i, cex = 0.5)
}
```