

Simulated Annealing (Recozimento Simulado)

UFMG

10/11/2025

Sumário

1 Simulação de Recozimento

- Introdução
- Processo de Recozimento
- Aplicabilidade
- Teoria
- Algoritmo Geral de Recozimento Simulado
- Implementação

Recozimento Simulado - Introdução

Também chamado de método meta-heurístico, este tem como intuito resolver problemas de otimização de grande complexidade. Este algoritmo foi introduzido por volta de 1980 por três pesquisadores, sendo eles: Kirkpatrick, Gelatt e Vecchi. O método traz soluções sub-ótimas, ou seja, eficientes e sem grande esforço computacional.

Tem como objetivo encontrar resultados satisfatórios, ainda que não exatos, sendo que muitas vezes estes resultados são inviáveis e/ou difíceis de serem alcançados.

Se enquadra na classe *Markov chain Monte Carlo* (MCMC) e, por ter caráter estocástico alinhado à sua capacidade de escapar de mínimos locais, este é um ótimo candidato para resolução de problemas complexos de otimização.

Processo de Recozimento

O conceito foi introduzido da ideia de recozimento, onde um sólido é levado para um estado de baixa energia após aumentar sua temperatura.

O processo consiste em duas etapas, sendo elas:

- "Derretimento" da sua estrutura ao levá-lo a uma temperatura muito elevada.
- Resfriamento esquematizado buscando atingir um estado sólido de energia mínima.

Uma vez em estado líquido as partículas do material exposto são distribuídas de forma aleatória.

O estado mínimo de energia só é alcançado com uma temperatura inicial que seja suficientemente alta e um tempo de resfriamento que seja logo o suficiente.

Aplicabilidade

O algoritmo **Simulated Annealing** tem aplicabilidade em várias áreas e aqui serão expostas algumas delas:

- Engenharia de software
- *Machine Learning*
- Teoria de filas
- Processos de manufatura
- logística e transporte com otimização

Dist. Weibull Tri-Paramétrica

O estudo será centrado em estimar os parâmetros primeiramente de uma distribuição Weibull com a seguinte função densidade:

$$f(x) = \frac{\beta}{\eta} \left(\frac{x-\gamma}{\eta} \right)^{\beta-1} e^{-\frac{x-\gamma}{\eta}^\beta}; \quad \beta > 0, \quad \eta > \gamma \geq 0. \quad (1)$$

Tenha ciência também que sua acumulada tem o seguinte resultado:

$$F(x) = 1 - e^{-\frac{x-\gamma}{\eta}^\beta},$$

sendo β , η , γ os parâmetros de forma, escala e locação, respectivamente.

A distribuição Weibull tri-paramétrica é raramente usada devido à dificuldade de se estimar estes parâmetros. Portanto, o uso de SA será feito para tentar estimar estes parâmetros. Aqui o algoritmo será fundamental para maximizar a função de verossimilhança.

Estimação por EMV

Temos ciência que a função de máxima verossimilhança é descrita da seguinte forma:

$$L(x) = \prod_{i=1}^n f_{x_i}(x_i; \theta)$$

Com isso, a log-verossimilhança é a seguinte:

$$\ln(L(x_1, \dots, x_n, \beta, \eta, \gamma)) = n \ln\left(\frac{\beta}{\eta}\right) + \sum_{i=1}^n \left(-\left(\frac{x_i - \gamma}{\eta}\right)^\beta + (\beta - 1) \ln\left(\frac{x_i - \gamma}{\eta}\right) \right). \quad (2)$$

Para encontrar as estimativas dos parâmetros devemos realizar as derivadas parciais e igualar a zero, da seguinte forma:

$$\frac{\partial}{\partial \theta} \ln(L(x)) = 0$$

Fica evidente que esta é uma função bem difícil de derivar, necessitando derivar gradiente ou mesmo aplicar métodos numéricos, portanto prosseguiremos para a ideia geral do algoritmo SA

Algoritmo

A principal vantagem deste método perante os demais é fato de que este, ao se empregar uma busca aleatória, **não se prende a pontos de mínimos locais**. O algoritmo aceita mudanças que aceitam a diminuição e também o aumento da função objetivo f . O seu aumento é aceito com probabilidade: $p = e^{-\frac{\Delta}{T}}$ sendo θ o aumento e T o parâmetro de controle, analogamente conhecido como temperatura do sistema. A sua implementação é simples:

- Uma representação de soluções possíveis,
- Um gerador de mudanças aleatórias nas soluções,
- Um meio de avaliar as funções do problema, e
- Um esquema de resfriamento (annealing schedule) – uma temperatura inicial e regras para diminuí-la à medida que a busca progride.

```
simulated_annealing <- function(f, S0, T0, L, alpha, T_min) {  
  T <- T0; S <- S0  
  S_star <- S0; f_S <- f(S)  
  f_S_star <- f_S  
  while (T > T_min) {  
    for (n in 1:L) {  
      S_n <- gerar_vizinho(S)  
      f_S_n <- f(S_n); Delta <- f_S_n - f_S  
      if (Delta <= 0) {  
        S <- S_n; f_S <- f_S_n  
      } else {  
        p <- exp(-Delta / T)  
        if (runif(1) < p) {  
          S <- S_n; f_S <- f_S_n  
        }}}}  
  if (f_S < f_S_star) {  
    S_star <- S; f_S_star <- f_S  
  }  
  T <- T * alpha  
}  
return(list(best_solution = S_star, best_cost = f_S_star))  
}
```


Código implementado

```
#bibliotecas
library(ggplot2); library(dplyr); library(tidyr); library(gridExtra)

#log-verossimilhança
loglik_weibull3 <- function(params, x) {
  b <- params[1]; g <- params[2]; c <- params[3]
  if (b <= 0 || g <= 0 || any(x <= c)) {
    return(-Inf)
  }
  z <- (x - c) / g
  ll <- length(x) * (log(b) - log(g)) + (b - 1) * sum(log(z)) - sum(z^b)
  return(ll)
}
```

```
# Gera vizinho (proposta) garantindo g>0, b>0, c < min(x) (senão rejeita)
propose_neighbor <- function(curr, x, sds = c(0.1, 0.1, 0.1)) {
  attempt <- 0
  repeat {
    attempt <- attempt + 1
    b1 <- curr[1] + rnorm(1, mean = 0, sd = sds[1])
    g1 <- curr[2] + rnorm(1, mean = 0, sd = sds[2])
    c1 <- curr[3] + rnorm(1, mean = 0, sd = sds[3])
    # enforce positivity of b,g by reflecting small negatives
    if (b1 <= 0) {
      b1 <- abs(b1) + 1e-6
    }
    if (g1 <= 0) {
      g1 <- abs(g1) + 1e-6
    }
    # require c1 < min(x) - tiny
    if (c1 < min(x) - 1e-8) {
      return(c(b1, g1, c1))
    }
    # else try again; after many attempts increase sd for c
    if (attempt > 50) {
      sds[3] <- sds[3] * 1.5
    }
  }
}
```

```
}

if (attempt > 1000) {
  # extremely unlikely; fallback: set c1 = min(x) - epsilon
  return(c(b1, g1, min(x) - 1e-6))
}

}

}

# SA algorithm (maximização da log-verossimilhança)
sa_weibull3 <- function(
  x,
  init = NULL,
  T0 = 100,
  Tf = 0.001,
  cooling_rate = 0.99,
  L = 5,
  sds = c(0.1, 0.1, 0.1),
  verbose = FALSE
) {
  n <- length(x)
  # initial solution: if not provided, use method of moments-ish guesses
  if (is.null(init)) {
    # use rweibull fit approximations: estimate c0 slightly below min(x)
```

```

c0 <- min(x) - 0.1 * sd(x)
if (c0 >= min(x)) {
  c0 <- min(x) - 1e-3
}
# fit two-parameter Weibull to x - c0 via linearization (quick guess)
y <- x - c0
y[y <= 0] <- min(y[y > 0]) * 0.1
# estimate using log-log regression
est <- try(
{
  ln_x <- log(y)
  ln_ln <- log(-log(1 - (1:n) / (n + 1)))
  fit <- lm(ln_x ~ ln_ln)
  b0 <- 1 / coef(fit)[2]
  g0 <- exp(coef(fit)[1])
  c(b0, g0, c0)
},
silent = TRUE
)
if (inherits(est, "try-error") || any(!is.finite(est))) {
  est <- c(1.5, sd(x), min(x) - 0.1)
}
curr <- est

```

```
    } else {
        curr <- init
    }

    curr_ll <- loglik_weibull3(curr, x)
    best <- curr
    best_ll <- curr_ll

    T <- T0
    history <- data.frame(eval = 1, ll = curr_ll)
    eval_count <- 1

    while (T > Tf) {
        ninner <- 1
        while (ninner <= L) {
            prop <- propose_neighbor(curr, x, sds = sds)
            prop_ll <- loglik_weibull3(prop, x)
            # If better, accept; else accept with probability exp((prop_ll - curr_ll) / T)
            if (prop_ll > curr_ll) {
                curr <- prop
                curr_ll <- prop_ll
            } else {
                delta <- prop_ll - curr_ll
```

```
if (runif(1) < exp(delta / T)) {
  curr <- prop
  curr_ll <- prop_ll
}
}
if (curr_ll > best_ll) {
  best <- curr
  best_ll <- curr_ll
}
eval_count <- eval_count + 1
history <- rbind(history, data.frame(eval = eval_count, ll = curr_ll)
ninner <- ninner + 1
}
T <- cooling_rate * T
}
return(list(
  best = best,
  best_ll = best_ll,
  history = history,
  evals = eval_count
))
}
```