Optimización con enjambre de partículas (*Particle Swarm Optimization*)

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Marzo, 2019

Tabla de contenidos

Versión PDF: Github

Introducción

La optimización por enjambre de partículas (*Particle Swarm Optimization, PSO*) es un método de optimización heurística orientado a encontrar mínimos o máximos globales. Su funcionamiento está inspirado en el comportamiento que tienen las bandadas de pájaros o bancos de peces en los que, el movimiento de cada individuo (dirección, velocidad, aceleración…), es el resultado de combinar las decisiones individuales de cada uno con el comportamiento del resto.

El método de enjambre de partículas es solo una de las muchas estrategias de optimización heurística que existen, una alternativa común son los algoritmos genéticos.

La optimización heurística no tiene por qué ser la forma de optimización más adecuada en todos los escenarios. Si el problema en cuestión puede optimizarse de forma analítica, suele ser más adecuado resolverlo de esta forma.

La implementación de algoritmo que se muestra en este documento pretende ser lo más explicativa posible aunque para ello no sea la más eficiente.

El código de las funciones desarrolladas a lo largo del documento puede descargarse en el siguiente link.

Algoritmo

Aunque existen variaciones, algunas de las cuales se describen a lo largo de este documento, en términos generales, la estructura de un algoritmo *PSO* para optimizar (maximizar o minimizar) una función con una o múltiples variables sigue los siguientes pasos:

---

1. Crear un enjambre inicial de n partículas. Cada partícula consta de 4 elementos: una posición que representa una determinada combinación de valores de las variables, el valor de la función objetivo en la posición donde se encuentra la partícula, una velocidad que indica cómo y hacia donde se desplaza la partícula, y un registro de la mejor posición en la que ha estado la partícula hasta el momento.
2. Evaluar cada partícula con la función objetivo.
3. Actualizar la posición y velocidad de cada partícula. Esta es la parte que proporciona al algoritmo la capacidad de optimización. En el apartado Mover partícula se describe con detalle el proceso.
4. Si no se cumple un criterio de parada, volver al paso 2.

En los siguientes apartados se implementan cada una de las etapas del proceso para, finalmente, combinarlas todas en una única función.

Crear partícula

Cada partícula está definida por una posición, velocidad y valor que varían a medida que la partícula se mueve. Además, también almacena la mejor posición en la que ha estado hasta el momento. Cuando se crea una nueva partícula, únicamente se dispone de información sobre su posición y velocidad (normalmente iniciada como cero), el resto de valores no se conocen hasta que la partícula es evaluada.

```
crear_particula <- function(n_variables,
                            limites_inf = NULL,
                            limites_sup = NULL,
                            verbose     = TRUE) {
  # Esta función crea una nueva partícula con una posición inicial definida por
  # una combinación de valores numéricos aleatorios y velocidad de 0. El rango de
  # posibles valores para cada variable (posición) puede estar acotado.
  #
  # ARGUMENTOS
  # ============================================================================
  # n_variables:   número de variables que definen la partícula.
  # limites_inf:   vector con el límite inferior de cada variable. Si solo se
  #                quiere imponer límites a algunas variables, emplear NA para
  #                las que no se quiere acotar.
  # limites_sup:   vector con el límite superior de cada variable. Si solo se
  #                quiere imponer límites a algunas variables, emplear NA para
  #                las que no se quieren acotar.
  # verbose:       mostrar información de la partícula creada.
  #
  # RETORNO
  # ============================================================================
  # Un objeto de tipo lista que representa la estructura de una partícula.
  # Al crear una nueva partícula, solo se dispone de información sobre su
  # posición inicial y velocidad, el resto de atributos están vacíos.
  #
  # posicion:       vector con la posición actual (valor de las variables) de la
  #                 partícula.
  # valor:          valor actual de la partícula. Resultado de evaluar la función
  #                 objetivo con la posición actual.
  # velocidad:      vector con la velocidad actual de la partícula.
  # mejor_valor:    mejor valor que ha tenido la partícula hasta el momento.
  # mejor_posicion: posición en la que la partícula ha tenido el mejor valor hasta
  #                 el momento.

  # Comprobaciones
  if (!is.null(limites_inf) & (length(limites_inf) != n_variables)) {
    stop(paste(
      "limites_inf debe tener un valor por cada variable.",
      "Si para alguna variable no se quiere límite, emplear NA.",
```

```
    "Ejemplo: limites_inf = c(10, NA, 10)"
  ))
} else if (!is.null(limites_sup) & length(limites_sup) != n_variables) {
  stop(paste(
    "limites_sup debe tener un valor por cada variable.",
    "Si para alguna variable no se quiere límite, emplear NA.",
    "Ejemplo: limites_sup = c(10, NA, 10)"
  ))
} else if (is.null(limites_sup) | is.null(limites_inf)) {
  warning(paste(
    "Es altamente recomendable indicar los límites dentro de los",
    "cuales debe buscarse la solución de cada variable.",
    "Por defecto se emplea [-10^3, 10^3]."
  ))
} else if (any(c(is.na(limites_sup), is.na(limites_inf)))) {
  warning(paste(
    "Los límites empleados por defecto cuando no se han definido son:",
    " [-10^3, 10^3]."
  ))
  cat("\n")
}

# Si no se especifica limites_inf, el valor mínimo que pueden tomar las variables
# es -10^3.
if (is.null(limites_inf)) {
  limites_inf <- rep(x = -10^3, times = n_variables)
}
# Si no se especifica limites_sup, el valor máximo que pueden tomar las variables
# es 10^3.
if (is.null(limites_sup)) {
  limites_sup <- rep(x = 10^3, times = n_variables)
}
# Si los límites no son nulos, se reemplazan aquellas posiciones NA por el valor
# por defecto -10^3 y 10^3.
if (!is.null(limites_inf)) {
  limites_inf[is.na(limites_inf)] <- -10^3
}
if (!is.null(limites_sup)) {
  limites_sup[is.na(limites_sup)] <- 10^3
}

# Se crea una lista que representa la partícula.
particula       <- vector(length = 5, mode = "list")
names(particula) <- c("posicion", "valor", "velocidad", "mejor_valor",
                      "mejor_posicion")


#Bucle para asignar un valor a cada una de las variables que definen la posición.
posicion <- rep(NA, times = n_variables)
```

```
  for (i in 1:n_variables) {
      # Para cada posición, se genera un valor aleatorio dentro del rango permitido
      # para esa variable.
      posicion[i] <- runif(n = 1, min = limites_inf[i], max = limites_sup[i])
  }
  particula[["posicion"]] <- posicion

  # La velocidad inicial de la partícula es 0.
  velocidad <- rep(0, times = n_variables)
  particula[["velocidad"]] <- velocidad

  if (verbose) {
    print("Nueva partícula creada")
    print("----------------------")
    print(paste(
      "Posición:",
      paste(particula[["posicion"]], collapse = ", ")
      ))
    print(paste(
      "Límites inferiores de cada variable:",
      paste(limites_inf, collapse = ", ")
    ))
    print(paste(
      "Límites superiores de cada variable:",
      paste(limites_sup, collapse = ", ")
    ))
    cat("\n")
  }

  return(particula)
}
```

Ejemplo

Se crea una partícula de longitud 2, con los valores de la primera variable acotados entre [-10, +10] y la segunda con [-10, NA].

```
particula <- crear_particula(
              n_variables  = 2,
              limites_inf  = c(-10, -10),
              limites_sup  = c(+10, NA),
              verbose      = TRUE
            )
```

```
##
## [1] "Nueva partícula creada"
## [1] "----------------------"
## [1] "Posición: 135.827860678546, -2.87671756930649"
## [1] "Límites inferiores de cada variable: -10, -10"
## [1] "Límites superiores de cada variable: -10, 1000"
```

Evaluar partícula

Evaluar una partícula consiste en calcular el valor de la función objetivo en la posición que ocupa la partícula es ese momento. Cada partícula almacena también la posición con mejor valor en la que ha estado hasta el momento. Para poder identificar si una nueva posición es mejor que las anteriores, es necesario conocer si se trata de un problema de minimización o maximización.

```r
evaluar_particula <- function(particula,
                              funcion_objetivo,
                              optimizacion,
                              verbose = TRUE) {


  # Esta función evalúa una partícula calculando el valor que toma la función
  # objetivo en la posición en la que se encuentra la partícula. Además, compara
  # si la nueva posición es mejor que las anteriores.
  #
  # ARGUMENTOS
  # ============================================================================
  # particula:        partícula que se quiere evaluar.
  # funcion_objetivo: función que se quiere optimizar.
  # optimizacion:     maximizar o minimizar. Dependiendo de esto, el mejor valor
  #                   histórico de la partícula será el mayor o el menor valor
  #                   que ha tenido hasta el momento.
  # verbose:          mostrar información del proceso por pantalla.
  #
  # RETORNO
  # ============================================================================
  # Devuelve una nueva partícula en la que los campos valor, mejor_valor y
  # mejor_posicion han sido actualizados.

  # Comprobaciones.
  if (!optimizacion %in% c("maximizar", "minimizar")) {
      stop("El argumento optimizacion debe ser: maximizar o minimizar")
  }

  # Se evalúa la partícula con la función objetivo.
```

```
  valor <- do.call(
              funcion_objetivo,
              args = as.list(particula[["posicion"]])
            )
  # Se almacena el nuevo valor.
  particula[["valor"]] <- valor

  # Se compara el valor actual con el mejor valor histórico. La comparación es
  # distinta dependiendo de si se desea maximizar o minimizar. Si no existe ningún
  # valor histórico, se almacena el actual. Si ya existe algún valor histórico se
  # compara con el actual y, de ser mayor este último, se sobrescribe.
  if (is.null(particula[["mejor_valor"]])) {
    particula[["mejor_valor"]]    <- particula[["valor"]]
    particula[["mejor_posicion"]] <- particula[["posicion"]]
  } else{
    if (optimizacion == "minimizar") {
      if (particula[["valor"]] < particula[["mejor_valor"]]) {
        particula[["mejor_valor"]]    <- particula[["valor"]]
        particula[["mejor_posicion"]] <- particula[["posicion"]]
      }
    } else {
      if (particula[["valor"]] > particula[["mejor_valor"]]) {
        particula[["mejor_valor"]]    <- particula[["valor"]]
        particula[["mejor_posicion"]] <- particula[["posicion"]]
      }
    }
  }

  if (verbose) {
    print(paste("La partícula ha sido evaluada"))
    print("----------------------------")
    print(paste("Valor actual:", particula[["valor"]]))
    cat("\n")
  }
  return(particula)
}
```

Ejemplo

Se crea y evalúa una partícula para el caso de minimización de la función $f(x_1, x_2) = x_1^2 + x_2^2$.

```
# Función objetivo a optimizar.
funcion <- function(x1, x2) {
             return(x1^2 + x2^2)
           }
```

```
particula_1 <- crear_particula(
                  n_variables  = 2,
                  limites_inf  = c(-10, -10),
                  limites_sup  = c(+10, +10),
                  verbose      = TRUE
                )
```

```
## [1] "Nueva partícula creada"
## [1] "----------------------"
## [1] "Posición: -2.01330345124006, 7.48544162604958"
## [1] "Límites inferiores de cada variable: -10, -10"
## [1] "Límites superiores de cada variable: 10, 10"
```

```
particula_1 <- evaluar_particula(
                  particula        = particula_1,
                  funcion_objetivo = funcion,
                  optimizacion     = "minimizar",
                  verbose          = TRUE
                )
```

```
## [1] "La partícula ha sido evaluada"
## [1] "----------------------------"
## [1] "Valor actual: 60.0852271237709"
```

Mover partícula

Mover una partícula implica actualizar su velocidad y posición. Este paso es el más importante ya que otorga al algoritmo la capacidad de optimizar.

La velocidad de cada partícula del enjambre se actualiza empleando la siguiente ecuación:

$$v_i(t+1) = wv_i(t) + c_1 r_1[\hat{x}_i\,(t) - x_i(t)] + c_2 r_2[g(t) - x_i(t)]$$

donde:

- $v_i(t+1)$: velocidad de la partícula i en el momento $t+1$, es decir, la nueva velocidad.
- $v_i(t)$: velocidad de la partícula i en el momento t, es decir, la velocidad actual.
- w : coeficiente de inercia, reduce o aumenta a la velocidad de la partícula.
- c_1 : coeficiente cognitivo.
- r_1 : vector de valores aleatorios entre 0 y 1 de longitud igual a la del vector velocidad.
- $\hat{x}_i\,(t)$: la mejor posición en la que ha estado la partícula i hasta el momento.
- $x_i(t)$: posición de la partícula i en el momento t.
- c_2 : coeficiente social.
- r_2 : vector de valores aleatorios entre 0 y 1 de longitud igual a la del vector velocidad.
- $g(t)$: mejor posición de todo el enjambre en el momento t, el mejor valor global.

Para comprender como se relaciona esta ecuación con el movimiento de la partícula, resulta útil diferenciar tres partes:

- $wv_i(t)$ es la componente de inercia, responsable de mantener a la partícula moviéndose en la dirección en la que lo ha estado haciendo hasta el momento. El valor recomendado del coeficiente de inercia w suele ser entre 0.8 y 1.2. Si $w < 1$, la partícula se va desacelerando a medida que avanzan las iteraciones, esto se traduce en menor exploración pero una convergencia hacia el óptimo más rápida. Si $w > 1$, la partícula se va acelerando, lo que permite explorar más zonas del espacio de la función pero dificulta la convergencia.

- $c_1 r_1[\hat{x}_i\,(t) - x_i(t)]$ es la componente cognitiva, responsable de que la partícula tienda a moverse hacia la posición donde ha obtenido mejores resultados hasta el momento. El coeficiente cognitivo c_1 suele estar acotado en el rango [0, 2], siendo 2 el valor recomendado. r_1 es un vector de valores aleatorios entre 0 y 1 (un valor por cada dimensión) que aporta cierto comportamiento estocástico al movimiento de las partículas, mejorando así la capacidad de escapar de mínimos locales.

- $c_2 r_2[g(t) - x_i(t)]$ es la componente social, responsable de que la partícula tienda a moverse hacia la mejor posición encontrada por el enjambre hasta el momento. Puede interpretarse como el "conocimiento colectivo". El valor del coeficiente social c_2 suele estar acotado en el rango [0, 2], siendo 2 el valor recomendado. r_2 es un vector de valores aleatorios entre 0 y 1 (un valor por cada dimensión) que aporta cierto comportamiento estocástico al movimiento de las partículas, mejorando así la capacidad de escapar de mínimos locales.

- La magnitud relativa entre la componente cognitiva y la componente social permite regular el comportamiento exploratorio del algoritmo. Cuanto mayor es el valor de c_1 respecto a c_2, mayor independencia de movimiento tiene cada partícula, lo que permite mayor exploración pero mayor lentitud en la convergencia. Por el contrario, cuanto mayor es el valor de c_2 respecto a c_1, más obligadas están las partículas a moverse hacia la mejor región encontrada hasta el momento, lo que reduce la exploración pero acelera la convergencia.
- En algunas versiones del algoritmo, r_1 y r_2 son escalares en lugar de vectores. Multiplicar cada componente de la velocidad por un valor aleatorio distinto añade mayores fluctuaciones al movimiento de las partículas, lo que, aun a riesgo de retrasar la convergencia, suele generar mejores resultados.

Una vez calculada la nueva velocidad, se puede actualizar la posición de la partícula con la ecuación:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

Uno de los principales problemas del algoritmo *PSO* es que las partículas suelen adquirir velocidades excesivamente altas, lo que les lleva a salirse de los límites del espacio de búsqueda o a que sean incapaces de converger en la región óptima. Es en este paso del algoritmo donde más investigaciones y adaptaciones se han hecho. Algunas de las soluciones son:

- Limitar la velocidad máxima que puede alcanzar una partícula. Siendo $[x_{min}, x_{max}]$ los límites inferior y superior del espacio de búsqueda de cada variable, la velocidad máxima que puede alcanzar la partícula en esa dirección es $v_{max} = k(x_{max} - x_{min})/2$, donde k suele ser un valor entre 0.1 y 1.

- Si el valor de alguna de las variables excede los límites impuestos, se sobrescribe con el valor del límite correspondiente y se reinicia su velocidad a cero.

- Reducción lineal del coeficiente de inercia w. Esta estrategia consiste en ir reduciendo el coeficiente de inercia a medida que avanzan las iteraciones. En las primeras iteraciones, las partículas tiene mucha capacidad de exploración y, a medida que avanza el proceso, va reduciéndose su velocidad favoreciendo la convergencia. Puede conseguirse este efecto con la ecuación:

$$w_t = (w_{max} - w_{min}) \frac{t_{max} - t}{t_{max}} + w_{min}$$

donde:

- w_t: coeficiente de inercia en la iteración t.
- w_{max}: coeficiente de inercia máximo. Valor con el que se inicia el algoritmo. Valor recomendado de 0.9.
- w_{min}: coeficiente de inercia mínimo. Valor que se alcanza en la última iteración. Valor recomendado de 0.4.
- t_{max}: número máximo de iteraciones.

La siguiente función actualiza la posición de una partícula teniendo en cuenta su posición y velocidad actual, la mejor posición global encontrada por el enjambre, los coeficientes de inercia, cognitivo, social y los límites de búsqueda.

```
mover_particula <- function(particula,
                            mejor_p_enjambre,
                            inercia        = 0.8,
                            peso_cognitivo = 2,
                            peso_social    = 2,
                            limites_inf    = NULL,
                            limites_sup    = NULL,
                            verbose        = TRUE) {
  # Esta función ejecuta el movimiento de una partícula, lo que implica actualizar
  # su velocidad y posición. Si se especifican límites, no se permite que la
  # partícula salga de la zona de búsqueda.
  #
  # ARGUMENTOS
  # ============================================================================
  # particula:        partícula que se quiere evaluar.
  # mejor_p_enjambre: mejor posición de todo el enjambre.
  # inercia:          coeficiente de inercia.
  # peso_cognitivo:   coeficiente cognitivo.
  # peso_social:      coeficiente social.
  # limites_inf:      vector con el límite inferior de cada variable. Si solo se
  #                   quiere imponer límites a algunas variables, emplear NA para
  #                   las que no se quiere acotar.
  # limites_sup:      vector con el límite superior de cada variable. Si solo se
```

```
#                   quiere imponer límites a algunas variables, emplear NA para
#                   las que no se quieren acotar.
# verbose:          mostrar información del proceso por pantalla.
#
# RETORNO
# ============================================================================
# Devuelve una nueva partícula en la que la posición y velocidad han sido
# actualizadas.

# Actualización de la velocidad.
componente_velocidad <- inercia * particula[["velocidad"]]
r1 <- runif(n = length(particula[["velocidad"]]), min = 0, max = 1)
r2 <- runif(n = length(particula[["velocidad"]]), min = 0, max = 1)
componente_cognitivo <- peso_cognitivo * r1 * (particula[["mejor_posicion"]] -
                                               particula[["posicion"]])
componente_social    <- peso_social * r2 * (mejor_p_enjambre -
                                            particula[["posicion"]])

nueva_velocidad      <- componente_velocidad +
                        componente_cognitivo +
                        componente_social

# Actualización de la posción.
nueva_posicion <- particula[["posicion"]] + nueva_velocidad

# Se comprueba si algún valor de la nueva posición supera los límites impuestos.
# En tal caso, se sobrescribe con el valor del límite correspondiente y se
# reinicia a 0 la velocidad de la partícula en esa componente.
for (i in seq_along(nueva_posicion)) {
  if (isTRUE(nueva_posicion[i] < limites_inf[i])) {
      nueva_posicion[i]  <- limites_inf[i]
      nueva_velocidad[i] <- 0
  }
  if (isTRUE(nueva_posicion[i] > limites_sup[i])) {
       nueva_posicion[i] <- limites_sup[i]
       nueva_velocidad[i] <- 0
  }
}

particula[["velocidad"]] <- nueva_velocidad
particula[["posicion"]]  <- nueva_posicion

if (verbose) {
  print(paste("La partícula se ha desplazado"))
  print("----------------------------")
  print(paste(
    "Nueva posición:",
    paste(particula[["posicion"]], collapse = ", ")
    ))
```

```
    cat("\n")
  }
  return(particula)
}
```

Ejemplo

```
particula_1 <- crear_particula(
                  n_variables  = 2,
                  limites_inf  = c(-10, -10),
                  limites_sup  = c(+10, 10),
                  verbose      = TRUE
                )
```

```
## [1] "Nueva partícula creada"
## [1] "----------------------"
## [1] "Posición: -6.96712323464453, 7.82502901740372"
## [1] "Límites inferiores de cada variable: -10, -10"
## [1] "Límites superiores de cada variable: 10, 10"
```

```
particula_1 <- evaluar_particula(
                  particula        = particula_1,
                  funcion_objetivo = funcion,
                  optimizacion     = "minimizar",
                  verbose          = TRUE
                )
```

```
## [1] "La partícula ha sido evaluada"
## [1] "----------------------------"
## [1] "Valor actual: 109.771885289934"
```

```
particula_1 <- mover_particula(
                  particula         = particula_1,
                  mejor_p_enjambre  = c(0, 0),
                  inercia           = 0.5,
                  peso_cognitivo    = 1,
                  peso_social       = 2,
                  limites_inf       = c(-10, -10),
                  limites_sup       = c(+10, +10),
                  verbose           = TRUE
                )
```

```
## [1] "La partícula se ha desplazado"
## [1] "----------------------------"
## [1] "Nueva posición: 6.79133890153902, -0.585360331251588"
```

Crear enjambre

La creación del enjambre consiste en la inicializar *n* nuevas partículas.

```
crear_enjambre <- function(n_particulas,
                           n_variables,
                           limites_inf = NULL,
                           limites_sup = NULL,
                           verbose     = TRUE) {
  # Esta función crea un enjambre con n partículas.
  #
  # ARGUMENTOS
  # ============================================================================
  # n_particulas;  número de partículas del enjambre.
  # n_variables:   número de variables que definen una partícula.
  # limites_inf:   vector con el límite inferior de cada variable. Si solo se
  #                quiere imponer límites a algunas variables, emplear NA para
  #                las que no se quiere acotar.
  # limites_sup:   vector con el límite superior de cada variable. Si solo se
  #                quiere imponer límites a algunas variables, emplear NA para
  #                las que no se quieren acotar.
  # verbose:       mostrar información del proceso por pantalla.
  #
  # RETORNO
  # ============================================================================
  # Un objeto lista con dos elementos:
  # particulas:      lista con todas las partículas del enjambre.
  # mejor_particula: la mejor partícula del enjambre.

  # Se crea la lista que representa el enjambre y los dos elementos que lo forman.
  enjambre        <- vector(length = 2, mode = "list")
  names(enjambre) <- c("particulas", "mejor_particula")
  enjambre[["particulas"]]        <- vector(length = n_particulas, mode = "list")
  names(enjambre[["particulas"]]) <- paste0("particula_", 1:n_particulas)

  # Al crear el enjambre, las partículas no han sido todavía evaluadas, por lo
  # no se conoce cuál es la mejor partícula.
  enjambre[["mejor_particula"]]   <- list()

  # Se crea cada una de las partículas y se almacenan en enjambre[["particulas"]]
  for (i in 1:n_particulas) {
    enjambre[["particulas"]][[i]] <- crear_particula(
                                       n_variables = n_variables,
                                       limites_inf = limites_inf,
                                       limites_sup = limites_sup,
                                       verbose     = verbose
                                     )
  }
```

```
  if (verbose) {
    print("Enjambre creado")
    print("---------------")
    print(paste("Número de partículas =", n_particulas))
    print(paste(
      "Límites inferiores de cada variable:",
      paste(limites_inf, collapse = ", ")
    ))
    print(paste(
      "Límites superiores de cada variable:",
      paste(limites_sup, collapse = ", ")
    ))
    cat("\n")
  }

  return(enjambre)
}
```

Ejemplo

Se crea un enjambre con 3 partículas en un espacio de 2 dimensiones.

```
enjambre <- crear_enjambre(
              n_particulas = 3,
              n_variables  = 2,
              limites_inf  = c(-100, -20),
              limites_sup  = c(+100, +20),
              verbose      = TRUE
            )
```

```
## [1] "Nueva partícula creada"
## [1] "----------------------"
## [1] "Posición: 65.4563085641712, -18.0680519808084"
## [1] "Límites inferiores de cada variable: -100, -20"
## [1] "Límites superiores de cada variable: 100, 20"
##
## [1] "Nueva partícula creada"
## [1] "----------------------"
## [1] "Posición: -12.6837724354118, 19.5982388686389"
## [1] "Límites inferiores de cada variable: -100, -20"
## [1] "Límites superiores de cada variable: 100, 20"
##
## [1] "Nueva partícula creada"
## [1] "----------------------"
## [1] "Posición: 8.10881000943482, -16.505682496354"
## [1] "Límites inferiores de cada variable: -100, -20"
## [1] "Límites superiores de cada variable: 100, 20"
##
```

```
## [1] "Enjambre creado"
## [1] "---------------"
## [1] "Número de partículas = 3"
## [1] "Límites inferiores de cada variable: -100, -20"
## [1] "Límites superiores de cada variable: 100, 20"
```

```
enjambre
```

```
## $particulas
## $particulas$particula_1
## $particulas$particula_1$posicion
## [1]  65.45631 -18.06805
##
## $particulas$particula_1$valor
## NULL
##
## $particulas$particula_1$velocidad
## [1] 0 0
##
## $particulas$particula_1$mejor_valor
## NULL
##
## $particulas$particula_1$mejor_posicion
## NULL
##
##
## $particulas$particula_2
## $particulas$particula_2$posicion
## [1] -12.68377  19.59824
##
## $particulas$particula_2$valor
## NULL
##
## $particulas$particula_2$velocidad
## [1] 0 0
##
## $particulas$particula_2$mejor_valor
## NULL
##
## $particulas$particula_2$mejor_posicion
## NULL
##
##
## $particulas$particula_3
## $particulas$particula_3$posicion
## [1]   8.10881 -16.50568
##
## $particulas$particula_3$valor
## NULL
##
```

```
## $particulas$particula_3$velocidad
## [1] 0 0
##
## $particulas$particula_3$mejor_valor
## NULL
##
## $particulas$particula_3$mejor_posicion
## NULL
##
##
##
## $mejor_particula
## list()
```

Evaluar enjambre

Evaluar un enjambre consiste en calcular el valor de la función objetivo en la posición de cada una de las partículas que lo forman.

```
evaluar_enjambre <- function(enjambre,
                             funcion_objetivo,
                             optimizacion,
                             verbose = TRUE) {

  # Esta función evalúa todas las partículas del enjambre, actualiza sus valores,
  # e identifica la mejor partícula.
  #
  # ARGUMENTOS
  # ============================================================================
  # enjambre:         enjambre que se quiere evaluar.
  # funcion_objetivo: función que se quiere optimizar.
  # optimizacion:     maximizar o minimizar. Dependiendo de esto, el mejor valor
  #                   histórico de la partícula será el mayor o el menor valor
  #                   que ha tenido hasta el momento.
  #
  # RETORNO
  # ============================================================================
  # Devuelve una nuevo enjambre en el que el valor de todas las partículas
  # y el de la mejor partícula ha sido actualizado.

  # Se evalúa cada partícula del enjambre.
  enjambre[["particulas"]] <- purrr::map(
                                .x               = enjambre[["particulas"]],
                                .f               = evaluar_particula,
                                funcion_objetivo = funcion_objetivo,
```

```
                                optimizacion     = optimizacion,
                                verbose          = verbose
                               )
  # Se identifica la mejor partícula de todo el enjambre. Si se está maximizando,
  # la mejor partícula es aquella con mayor valor. Lo contrario si se está
  # minimizando.

  # Se selecciona inicialmente como mejor partícula la primera.
  mejor_particula <- enjambre[["particulas"]][[1]]

  # Se comparan todas las partículas del enjambre.
  for (i in seq_along(enjambre[["particulas"]])) {
    if (optimizacion == "minimizar") {
      if (enjambre[["particulas"]][[i]][["valor"]] < mejor_particula[["valor"]]) {
        mejor_particula <- enjambre[["particulas"]][[i]]
      }
    } else {
      if (enjambre[["particulas"]][[i]][["valor"]] > mejor_particula[["valor"]]) {
        mejor_particula <- enjambre[["particulas"]][[i]]
      }
    }
  }

  # Se almacena la mejor partícula en enjambre[["mejor_particula"]].
  enjambre[["mejor_particula"]] <- mejor_particula

  if (verbose) {
    print("Enjambre evaluado")
    print("-----------------")
    print(paste(
      "Mejor posición encontrada:",
       paste(mejor_particula[["posicion"]], collapse = ", ")
    ))
    print(paste(
      "Mejor valor encontrado:",
       mejor_particula[["valor"]]
    ))
    cat("\n")
  }
  return(enjambre)
}
```

Ejemplo

Se evalúa el enjambre para el caso de minimización de la función $f(x_1, x_2) = x_1^2 + x_2^2$.

```
# Función objetivo a optimizar.
funcion <- function(x1, x2) {
             return(x1^2 + x2^2)
           }

enjambre <- crear_enjambre(
              n_particulas = 3,
              n_variables  = 2,
              limites_inf  = c(-10, -10),
              limites_sup  = c(+10, +10),
              verbose      = TRUE
            )
```

```
## [1] "Nueva partícula creada"
## [1] "----------------------"
## [1] "Posición: 6.69601168483496, -1.97036528494209"
## [1] "Límites inferiores de cada variable: -10, -10"
## [1] "Límites superiores de cada variable: 10, 10"
##
## [1] "Nueva partícula creada"
## [1] "----------------------"
## [1] "Posición: 2.80870319344103, 6.54759038705379"
## [1] "Límites inferiores de cada variable: -10, -10"
## [1] "Límites superiores de cada variable: 10, 10"
##
## [1] "Nueva partícula creada"
## [1] "----------------------"
## [1] "Posición: -9.54163048416376, -6.67051671072841"
## [1] "Límites inferiores de cada variable: -10, -10"
## [1] "Límites superiores de cada variable: 10, 10"
##
## [1] "Enjambre creado"
## [1] "---------------"
## [1] "Número de partículas = 3"
## [1] "Límites inferiores de cada variable: -10, -10"
## [1] "Límites superiores de cada variable: 10, 10"
```

```
enjambre <- evaluar_enjambre(
              enjambre         = enjambre,
              funcion_objetivo = funcion,
              optimizacion     = "minimizar",
              verbose          = TRUE
            )
```

```
## [1] "La partícula ha sido evaluada"
## [1] "----------------------------"
## [1] "Valor actual: 48.7189118395512"
##
## [1] "La partícula ha sido evaluada"
## [1] "----------------------------"
## [1] "Valor actual: 50.759753505485"
##
## [1] "La partícula ha sido evaluada"
## [1] "----------------------------"
## [1] "Valor actual: 135.53850548443"
##
## [1] "Enjambre evaluado"
## [1] "-----------------"
## [1] "Mejor posición encontrada: 6.69601168483496, -1.97036528494209"
## [1] "Mejor valor encontrado: 48.7189118395512"
```

Mover enjambre

Mover el enjambre consiste en actualizar la posición de cada una de las partículas que lo forman.

```
mover_enjambre <- function(enjambre,
                           inercia        = 0.8,
                           peso_cognitivo = 2,
                           peso_social    = 2,
                           limites_inf    = NULL,
                           limites_sup    = NULL,
                           verbose        = TRUE) {
  # Esta función mueve todas las partículas del enjambre.
  #
  # ARGUMENTOS
  # ============================================================================
  # enjambre:       enjambre que se quiere mover.
  # optimizacion:   si se desea maximizar o minimizar la función.
  # inercia:        coeficiente de inercia.
  # peso_cognitivo: coeficiente cognitivo.
  # peso_social:    coeficiente social.
  # limites_inf:    vector con el límite inferior de cada variable. Si solo se
  #                 quiere imponer límites a algunas variables, emplear NA para
  #                 las que no se quiere acotar.
  # limites_sup:    vector con el límite superior de cada variable. Si solo se
  #                 quiere imponer límites a algunas variables, emplear NA para
  #                 las que no se quieren acotar.
  # verbose:        mostrar información del proceso por pantalla.
  #
  # RETORNO
  # ============================================================================
  # Devuelve una nuevo enjambre en el que la posición de todas las partículas
  # ha sido actualizada.

  # Se actualiza la posición de cada una de las partículas que forman el enjambre.
  mejor_posicion_enjambre <- enjambre[["mejor_particula"]][["posicion"]]
  enjambre[["particulas"]] <- purrr::map(
                                .x = enjambre[["particulas"]],
                                .f = mover_particula,
                                mejor_p_enjambre  = mejor_posicion_enjambre,
                                inercia           = inercia,
                                peso_cognitivo    = peso_cognitivo,
                                peso_social       = peso_social,
                                limites_inf       = limites_inf,
                                limites_sup       = limites_sup,
                                verbose           = verbose
                              )
```

```
  if (verbose){
    print("El la posición de todas las partículas del enjambre ha sido
actualizada.")
    cat("\n")
  }
  return(enjambre)
}
```

Ejemplo

```
# Función objetivo a optimizar.
funcion <- function(x1, x2) {
  return(x1^2 + x2^2)
}

enjambre <- crear_enjambre(
              n_particulas = 2,
              n_variables  = 2,
              limites_inf  = c(-10, -10),
              limites_sup  = c(+10, +10),
              verbose      = FALSE
            )

enjambre <- evaluar_enjambre(
              enjambre         = enjambre,
              funcion_objetivo = funcion,
              optimizacion     = "minimizar",
              verbose          = TRUE)
```

```
## [1] "La partícula ha sido evaluada"
## [1] "----------------------------"
## [1] "Valor actual: 143.101945991273"
##
## [1] "La partícula ha sido evaluada"
## [1] "----------------------------"
## [1] "Valor actual: 43.4363111385841"
##
## [1] "Enjambre evaluado"
## [1] "-----------------"
## [1] "Mejor posición encontrada: 6.54304390773177, 0.790498298592865"
## [1] "Mejor valor encontrado: 43.4363111385841"
```

```
enjambre <- mover_enjambre(
              enjambre       = enjambre,
              inercia        = 0.5,
              peso_cognitivo = 1,
              peso_social    = 2,
              limites_inf    = c(-10, -10),
              limites_sup    = c(+10, +10),
              verbose        = TRUE
            )
```

```
## [1] "La partícula se ha desplazado"
## [1] "----------------------------"
## [1] "Nueva posición: 10, 6.24950640189017"
##
## [1] "La partícula se ha desplazado"
## [1] "----------------------------"
## [1] "Nueva posición: 6.54304390773177, 0.790498298592865"
##
## [1] "La posición de todas las partículas del enjambre ha sido actualizada."
```

Algoritmo completo

```
optimizar_enjambre <- function(
                        funcion_objetivo,
                        n_variables,
                        optimizacion,
                        limites_inf        = NULL,
                        limites_sup        = NULL,
                        n_particulas       = 100,
                        n_iteraciones      = 50,
                        inercia            = 0.8,
                        reduc_inercia      = TRUE,
                        inercia_max        = 0.9,
                        inercia_min        = 0.4,
                        peso_cognitivo     = 2,
                        peso_social        = 2,
                        parada_temprana    = FALSE,
                        rondas_parada      = NULL,
                        tolerancia_parada  = NULL,
                        verbose            = FALSE,
                        ...) {
  # ARGUMENTOS
  # ============================================================================
  # funcion_objetivo: nombre de la función que se desea optimizar. Debe de haber
  #                   sido definida previamente.
  # n_variables:      número de variables que definen una partícula.
  # optimizacion:     "maximizar" o "minimizar".
  # limites_inf:      vector con el límite inferior de cada variable. Si solo se
  #                   quiere imponer límites a algunas variables, emplear NA para
  #                   las que no se quiere acotar.
  # limites_sup:      vector con el límite superior de cada variable. Si solo se
  #                   quiere imponer límites a algunas variables, emplear NA para
  #                   las que no se quieren acotar.
  # n_particulas:     número total de partículas del enjambre.
  # n_iteraciones:    número total de iteraciones de optimización.
  # inercia:          coeficiente de inercia.
  # peso_cognitivo:   coeficiente cognitivo.
  # peso_social:      coeficiente social.
  # reduc_inercia:    activar la reducción del coeficiente de inercia. En tal caso,
  #                   el argumento inercia es ignorado.
  # inercia_max       valor inicial del coeficiente de inercia si se activa
  #                   reduc_inercia.
  # inercia_min       valor minimo del coeficiente de inercia si se activa
  #                   reduc_min.
  # parada_temprana:  si durante las últimas "rondas_parada" generaciones la
  #                   diferencia absoluta entre mejores individuos no es superior
  #                   al valor de "tolerancia_parada", se detiene el algoritmo y no
  #                   se crean nuevas generaciones.
```

```
# rondas_parada:    número de generaciones consecutivas sin mejora mínima para
#                   que se active la parada temprana.
# tolerancia_parada: valor mínimo que debe tener la diferencia de generaciones
#                    consecutivas para considerar que hay cambio.
# verbose:          TRUE para que se imprima por pantalla el resultado de cada
#                   paso del algoritmo.

# RETORNO
# ============================================================================
# La función devuelve una lista con 4 elementos:
# historico_enjambre: información sobre el enjambre en cada iteración.
# optim_valor:        valor óptimo encontrado.
# optim_posicion:     posición donde se ha encontrado el valor óptimo.
# resultados_df:      data.frame con la información del mejor valor y posición
#                     encontrado en cada iteración, así como la mejora respecto
#                     a la iteración anterior.


# LIBRERÍAS NECESARIAS
# ============================================================================
library(purrr)

# COMPROBACIONES INICIALES
# ============================================================================
# Si se activa la parada temprana, hay que especificar los argumentos
# rondas_parada y tolerancia_parada.


if (isTRUE(parada_temprana) &
    (is.null(rondas_parada) | is.null(tolerancia_parada)) ) {
  stop(paste(
    "Para activar la parada temprana es necesario indicar un valor",
    "de rondas_parada y de tolerancia_parada."
  ))
}

# Si se activa la reducción de inercia, hay que especificar los argumentos
# inercia_max y inercia_min.
if (isTRUE(reduc_inercia) &
    (is.null(inercia_max) | is.null(inercia_min)) ) {
  stop(paste(
    "Para activar la reducción de inercia es necesario indicar un valor",
    "de inercia_max y de inercia_min."
  ))
}

# ESTABLECER LOS LÍMITES DE BÚSQUEDA SI EL USUARIO NO LO HA HECHO
# ============================================================================
if (is.null(limites_sup) | is.null(limites_inf)) {
  warning(paste(
```

```
    "Es altamente recomendable indicar los límites dentro de los",
    "cuales debe buscarse la solución de cada variable.",
    "Por defecto se emplea: [-10^3, 10^3]."
  ))
} else if (any(c(is.na(limites_sup), is.na(limites_inf)))) {
  warning(paste(
    "Los límites empleados por defecto cuando no se han definido son:",
    " [-10^3, 10^3]."
  ))
  cat("\n")
}

# Si no se especifica limites_inf, el valor mínimo que pueden tomar las variables
# es -10^3.
if (is.null(limites_inf)) {
  limites_inf <- rep(x = -10^3, times = n_variables)
}
# Si no se especifica limites_sup, el valor máximo que pueden tomar las variables
# es 10^3.
if (is.null(limites_sup)) {
  limites_sup <- rep(x = 10^3, times = n_variables)
}
# Si los límites no son nulos, se reemplazan aquellas posiciones NA por el valor
# por defecto -10^3 y 10^3.
if (!is.null(limites_inf)) {
  limites_inf[is.na(limites_inf)] <- -10^3
}
if (!is.null(limites_sup)) {
  limites_sup[is.na(limites_sup)] <- 10^3
}

# ALMACENAMIENTO DE RESULTADOS
# ==============================================================================
# Por cada iteración se almacena: todo el enjambre, el mejor valor y la mejor
# posición encontradas en la iteración, y la diferencia respecto al mejor valor
# de la iteración anterior.
historico_enjambre        <- vector(mode = "list", length = n_iteraciones)
names(historico_enjambre) <- paste("iter", 1:n_iteraciones, sep = "_")
mejor_valor_enjambre      <- rep(NA, times = n_iteraciones)
mejor_posicion_enjambre   <- rep(NA, times = n_iteraciones)
diferencia_abs            <- rep(NA, times = n_iteraciones)


# CREACIÓN DEL ENJAMBRE INICIAL
# ==============================================================================
enjambre <- crear_enjambre(
              n_particulas = n_particulas,
              n_variables  = n_variables,
              limites_inf  = limites_inf,
              limites_sup  = limites_sup,
```

```
                verbose      = verbose
              )

  # ITERACIÓNES
  # ============================================================================
  for (i in 1:n_iteraciones) {
    if (verbose) {
      print("---------------------")
      print(paste("Iteración:", i))
      print("---------------------")
    }

    # EVALUAR PARTÍCULAS DEL ENJAMBRE
    # ============================================================================
    enjambre <- evaluar_enjambre(
                  enjambre         = enjambre,
                  funcion_objetivo = funcion,
                  optimizacion     = optimizacion,
                  verbose          = verbose
                )
    # SE ALMACENA EL ENJAMBRE Y LA MEJOR SOLUCIÓN ENCONTRADA EN ÉL
    # ============================================================================
    historico_enjambre[[i]]      <- enjambre
    mejor_valor_enjambre[[i]]    <- enjambre[["mejor_particula"]][["valor"]]
    mejor_posicion_enjambre[[i]] <- paste(
                                      enjambre[["mejor_particula"]][["posicion"]],
                                      collapse = ", "
                                    )
    # SE CALCULA LA DIFERENCIA ABSOLUTA RESPECTO A LA ITERACION ANTERIOR
    # ============================================================================
    # La diferencia solo puede calcularse a partir de la segunda generación.
    if (i > 1) {
      diferencia_abs[[i]] <- abs(mejor_valor_enjambre[[i-1]]-mejor_valor_enjambre[[i]])
    }

    # CRITERIO DE PARADA
    # ============================================================================
    # Si durante las últimas n generaciones, la diferencia absoluta entre mejores
    # partículas no es superior al valor de tolerancia_parada, se detiene el
    # algoritmo y no se crean nuevas generaciones.
    if (parada_temprana && (i > rondas_parada)) {
      ultimos_n <- tail(diferencia_abs[!is.na(diferencia_abs)], n = rondas_parada)
      if (all(ultimos_n < tolerancia_parada)) {
        print(paste(
          "Algoritmo detenido en la iteración", i,
          "por falta cambio absoluto mínimo de", tolerancia_parada,
          "durante", rondas_parada,
          "iteraciones consecutivas."
        ))
        break()
```

```
      }
    }
    # MOVER PARTÍCULAS DEL ENJAMBRE
    # ============================================================================
    # Si se ha activado la reducción de inercia, se recalcula su valor para la
    # iteración actual.
    if (isTRUE(reduc_inercia)) {
      inercia <- (inercia_max-inercia_min)*(n_iteraciones-i)/n_iteraciones+inercia_min
    }

    enjambre <- mover_enjambre(
                  enjambre       = enjambre,
                  inercia        = inercia,
                  peso_cognitivo = peso_cognitivo,
                  peso_social    = peso_social,
                  limites_inf    = limites_inf,
                  limites_sup    = limites_sup,
                  verbose        = verbose
                )
  }
  # IDENTIFICACIÓN DEL MEJOR INDIVIDUO DE TODO EL PROCESO
  # ============================================================================
   optim_valor    <- min(mejor_valor_enjambre, na.rm = TRUE)
   optim_posicion <- mejor_posicion_enjambre[which.min(mejor_valor_enjambre)]
   optim_posicion <- as.numeric(unlist(strsplit(x = optim_posicion, split = ", ")))

  # RESULTADOS
  # ============================================================================
  # Se crea un dataframe con un resumen de la información de cada iteración.
  resultados_df <- data.frame(
                     iteracion               = 1:n_iteraciones,
                     mejor_valor_enjambre    = mejor_valor_enjambre,
                     mejor_posicion_enjambre = mejor_posicion_enjambre,
                     diferencia_abs          = diferencia_abs
                   )
  # Si el algoritmo se ha detenido de forma temprana, se eliminan las iteraciones
  # vacías de resultados_df.
  resultados_df <- resultados_df[!is.na(mejor_valor_enjambre), ]

  return(
    list(historico_enjambre = historico_enjambre,
         optim_valor        = optim_valor,
         optim_posicion     = optim_posicion,
         resultados_df      = resultados_df
         )
  )
}
```

Ejemplo 1

En este ejemplo se pretende evaluar la capacidad del algoritmo genético para encontrar el mínimo de la función $f(x_1, x_2) = x_1^2 + x_2^2$. El mínimo global de esta función puede obtenerse de forma analítica igualando las derivadas parciales a cero, lo que permite comparar el resultado obtenido.

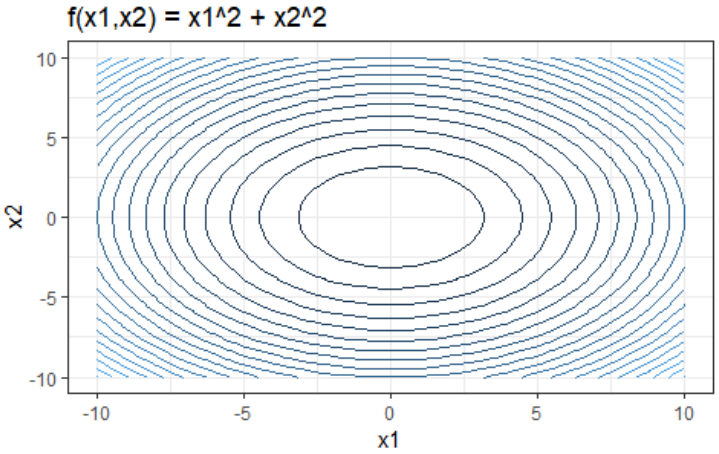
$$f(x_1 = 0, x_2 = 0) = 0$$

Función objetivo

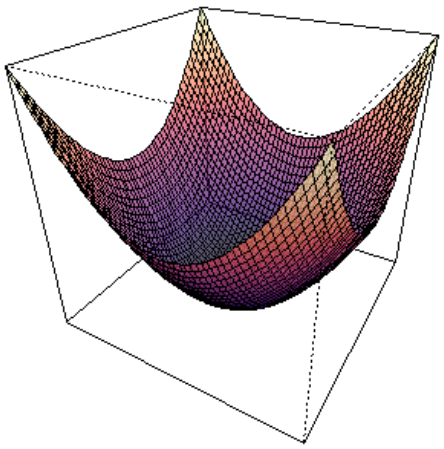
```
# Función objetivo a optimizar.
funcion <- function(x1, x2){
             return(x1^2 + x2^2)
           }
```

Representación gráfica de la función.

```
library(tidyverse)
x1 <- seq(-10, 10, length.out = 50)
x2 <- seq(-10, 10, length.out = 50)
datos <- expand.grid(x1 = x1, x2 = x2)
datos <- datos %>%
         mutate(f_x = map2_dbl(x1, x2, .f = funcion))
ggplot(data = datos, aes(x = x1, y = x2, z = f_x)) +
  geom_contour(aes(colour = stat(level)), bins = 20) +
  labs(title = "f(x1,x2) = x1^2 + x2^2") +
  theme_bw() +
  theme(legend.position = "none")
```



```
x1  <- seq(-10, 10, length.out = 50)
x2  <- seq(-10, 10, length.out = 50)
f_x <- outer(x1,x2, FUN = funcion)

library(viridis)
colores        <- viridis::magma(n = 100, alpha = 0.7)
z.facet.center <- (f_x[-1, -1] + f_x[-1, -ncol(f_x)] +
                     f_x[-nrow(f_x), -1] +
                     f_x[-nrow(f_x), -ncol(f_x)])/4
z.facet.range  <- cut(z.facet.center, 100)

par(mai = c(0,0,0,0))
persp(x = x1, y = x2, z = f_x, shade = 0.8,  phi = 30,
      theta = 30, col = colores[z.facet.range], axes = FALSE)
```



Optimización

```
optimizacion <- optimizar_enjambre(
                  funcion_objetivo = funcion,
                  n_variables      = 2,
                  optimizacion     = "minimizar",
                  limites_inf      = c(-10, -10),
                  limites_sup      = c(+10, +10),
                  n_particulas     = 100,
                  n_iteraciones    = 250,
                  inercia          = 0.8,
                  reduc_inercia    = TRUE,
                  inercia_max      = 0.9,
                  inercia_min      = 0.4,
                  peso_cognitivo   = 1,
                  peso_social      = 2,
```

```
                 parada_temprana  = TRUE,
                 rondas_parada    = 5,
                 tolerancia_parada = 10^-8,
                 verbose          = FALSE
                )
```

```
## [1] "Algoritmo detenido en la iteración 137 por falta cambio absoluto mínimo de
1e-08 durante 5 iteraciones consecutivas."
```

Resultados

El objeto devuelto por la función `optimizar_enjambre()` almacena la información (valor, posición,...) de las partículas del enjambre en cada iteración.

```
head(optimizacion$resultados_df)
```

```
##     iteracion mejor_valor_enjambre
## 1           1         1.287999e+00
## 2           2         6.701285e-02
## 3           3         8.970928e-02
## 4           4         2.430545e-02
## 5           5         1.011040e-01
## 6           6         3.412195e-02
## 7           7         3.051235e-02
## 8           8         1.257731e-01
## 9           9         3.112394e-02
## 10         10         6.708297e-02
```

```
##                          mejor_posicion_enjambre diferencia_abs
## 1           0.309223313815892, 1.09196168836206             NA
## 2          0.159781031688139, -0.203673434870637   1.220987e+00
## 3          0.168864270121705, -0.247374487538312   2.269643e-02
## 4          -0.103611974968279, 0.116490385829586   6.540383e-02
## 5          -0.289866265667437, 0.130696584052391   7.679860e-02
## 6          0.0655160194758426, 0.172712477037538   6.698210e-02
## 7         0.174012013909475, -0.0152372126241203   3.609595e-03
## 8           0.230864112593336, 0.269211583100114   9.526076e-02
## 9           0.0125281087809195, 0.17597438406977   9.464918e-02
## 10         0.0279445891197634, 0.257491888557294   3.595904e-02
```

Mejor individuo

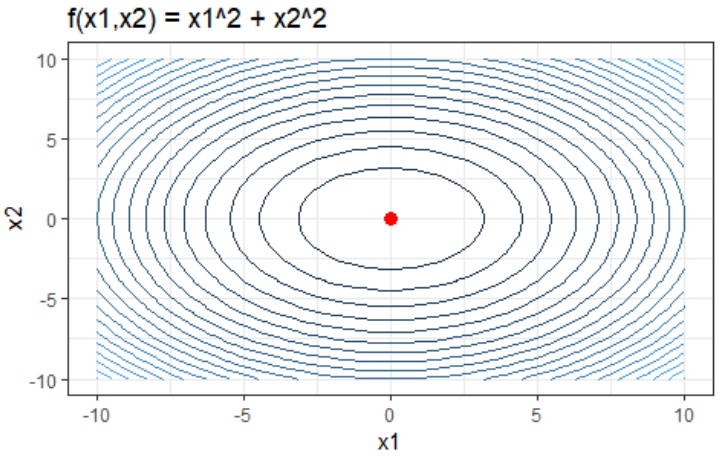
```
optimizacion$optim_posicion
```

```
## [1] 2.095577e-05 1.698586e-05
```

```
optimizacion$optim_valor
```

```
## [1] 7.276638e-10
```

```
ggplot(data = datos, aes(x = x1, y = x2, z = f_x)) +
  geom_contour(aes(colour = stat(level)), bins = 20) +
  annotate(
    geom = "point",
    x = optimizacion$optim_posicion[1],
    y = optimizacion$optim_posicion[2],
    color = "red",
    size = 3
  ) +
  labs(title = "f(x1,x2) = x1^2 + x2^2") +
  theme_bw() +
  theme(legend.position = "none")
```



Evolución del enjambre

En el siguiente gráfico se puede ver cómo evoluciona el valor de la mejor partícula del enjambre a medida que avanzan las iteraciones. Se excluyen las dos primeras iteraciones para centrar la gráfica.

```
ggplot(data = optimizacion$resultados_df %>% filter(iteracion > 2),
       aes(x = iteracion, y = mejor_valor_enjambre)) +
  geom_line(aes(group = 1)) +
  geom_point() +
  labs(title = "Evolución del valor de la mejor partícula del enjambre") +
  theme_bw() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank())
```

Animación de cómo avanza la búsqueda del mínimo (para formato HTML)

```
library(gganimate)

# Se extrae la posición de cada partícula en cada iteración.
extraer_particulas <- function(enjambre){
                        particulas <- enjambre[["particulas"]]
                        particulas <- map(.x = particulas,
                                          .f = function(x){x[["posicion"]]})
                        return(particulas)
                      }

particulas <- enframe(optimizacion$historico_enjambre,
                      name = "iteracion",
                      value = "enjambre") %>%
              mutate(particulas = map(enjambre, .f = extraer_particulas)) %>%
              select(-enjambre) %>%
              unnest() %>%
              mutate(x1 = map_dbl(particulas, .f = function(x){x[1]}),
                     x2 = map_dbl(particulas, .f = function(x){x[2]})) %>%
```

```
              select(-particulas) %>%
              mutate(iteracion = stringr::str_remove(iteracion, "iter_"),
                     iteracion = as.numeric(iteracion))

gift <- ggplot() +
        geom_contour(data = datos,
                     aes(x = x1, y = x2, z = f_x, colour = stat(level)),
                     bins = 20) +
        geom_point(data = particulas,
                   aes(x = x1, y = x2),
                   color = "red",
                   size = 1.8) +
        theme_bw() +
        theme(legend.position = "none") +
        transition_manual(frames = iteracion) +
        ggtitle("Posición del mínimo encontrado",
                subtitle = "Generación {frame}")

animate(plot = gift, fps = 3)
```

Ejemplo 2

En este ejemplo se pretende evaluar la capacidad del algoritmo genético para encontrar el mínimo de la función de *Mishra Bird*.

$$f(x_1, x_2) = sin(x_2)exp(1 - cos(x_1))^2 + cos(x_1)exp(1 - sin(x_2))^2 + (x_1 - x_2)^2$$

Para la región acotada entre:

$$-10 \leq x_1 \leq 0$$

$$-6.5 \leq x_2 \leq 0$$

la función tiene múltiples mínimos locales y un único el mínimo global que se encuentra en:

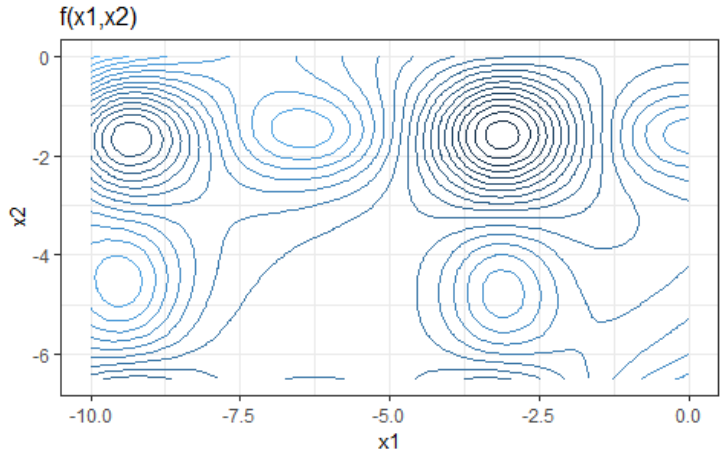
$$f(-3.1302468, -1.5821422) = -106.7645367$$

Función objetivo

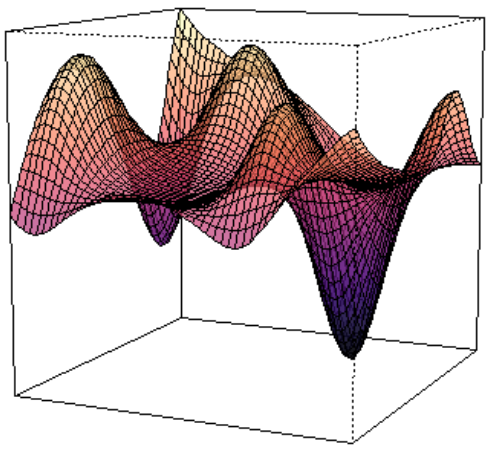
```
# Función objetivo a optimizar.
funcion <- function(x1, x2){
            sin(x2)*exp(1-cos(x1))^2 + cos(x1)*exp(1-sin(x2))^2 + (x1-x2)^2
          }
```

Representación gráfica de la función.

```
x1 <- seq(-10, 0, length.out = 100)
x2 <- seq(-6.5, 0, length.out = 100)

datos <- expand.grid(x1 = x1, x2 = x2)
datos <- datos %>%
         mutate(f_x = map2_dbl(x1, x2, .f = funcion))

ggplot(data = datos, aes(x = x1, y = x2, z = f_x)) +
  geom_contour(aes(colour = stat(level)), bins = 20) +
  labs(title = "f(x1,x2)") +
  theme_bw() +
  theme(legend.position = "none",
        title = element_text(size = 10))
```




```
x1  <- seq(-10, 0, length.out = 50)
x2  <- seq(-6.5, 0, length.out = 50)
f_x <- outer(x1, x2, FUN = funcion)

library(viridis)
colores        <- viridis::magma(n = 100, alpha = 0.7)
z.facet.center <- (f_x[-1, -1] + f_x[-1, -ncol(f_x)] +
                     f_x[-nrow(f_x), -1] +
                     f_x[-nrow(f_x), -ncol(f_x)])/4
z.facet.range  <- cut(z.facet.center, 100)

par(mai = c(0,0,0,0))
persp(x = x1, y = x2, z = f_x,
      shade = 0.8,
      r     = 8,
      phi   = 10,
      theta = 25,
      col   = colores[z.facet.range],
      axes  = FALSE)
```



Optimización

```
optimizacion <- optimizar_enjambre(
                funcion_objetivo = funcion,
                n_variables      = 2,
                optimizacion     = "minimizar",
                limites_inf      = c(-10, -6.5),
                limites_sup      = c(0, 0),
                n_particulas     = 100,
                n_iteraciones    = 250,
                inercia          = 0.8,
                reduc_inercia    = TRUE,
                inercia_max      = 0.9,
                inercia_min      = 0.4,
                peso_cognitivo   = 2,
                peso_social      = 2,
                parada_temprana  = TRUE,
                rondas_parada    = 5,
                tolerancia_parada = 10^-8,
                verbose          = FALSE
               )
```

```
## [1] "Algoritmo detenido en la iteración 151 por falta cambio absoluto mínimo de
1e-08 durante 5 iteraciones consecutivas."
```

Resultados

El objeto devuelto por la función `optimizar_enjambre()` almacena la información (valor, posición,...) de las partículas del enjambre en cada iteración.

```
head(optimizacion$resultados_df)
```

```
##     iteracion mejor_valor_enjambre              mejor_posicion_enjambre
## 1           1             -93.67964  -2.9970263550058, -1.98263388394844
## 2           2            -106.47412 -3.06948979624971, -1.55812320999118
## 3           3            -105.59291 -3.00733857905977, -1.55342705236961
## 4           4            -103.88333 -3.29958661889437, -1.52763975271546
## 5           5            -105.89360 -3.17144327499908, -1.68228800705677
## 6           6            -104.02750 -2.95740268139987, -1.66900547919474
## 7           7            -104.58627 -3.00253318608191, -1.70011698766784
## 8           8            -106.50933 -3.07809294208485, -1.62602187287149
## 9           9            -104.33501 -3.05424170109118, -1.43006939672311
## 10         10            -105.41181 -3.22526515701104, -1.51229995040412
##     diferencia_abs
```

```
## 1                NA
## 2     1.279447e+01
## 3     8.812069e-01
## 4     1.709580e+00
## 5     2.010273e+00
## 6     1.866101e+00
## 7     5.587713e-01
## 8     1.923056e+00
## 9     2.174321e+00
## 10    1.076806e+00
```

Mejor individuo

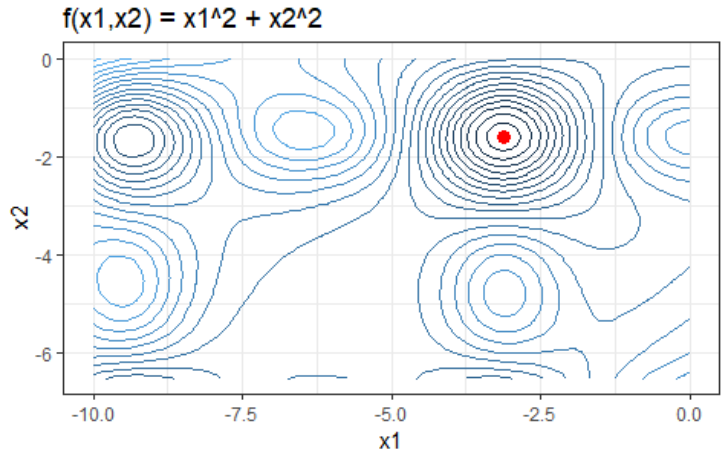
```
optimizacion$optim_posicion
```

```
## [1] -3.122844 -1.589536
```

```
optimizacion$optim_valor
```

```
## [1] -106.7877
```

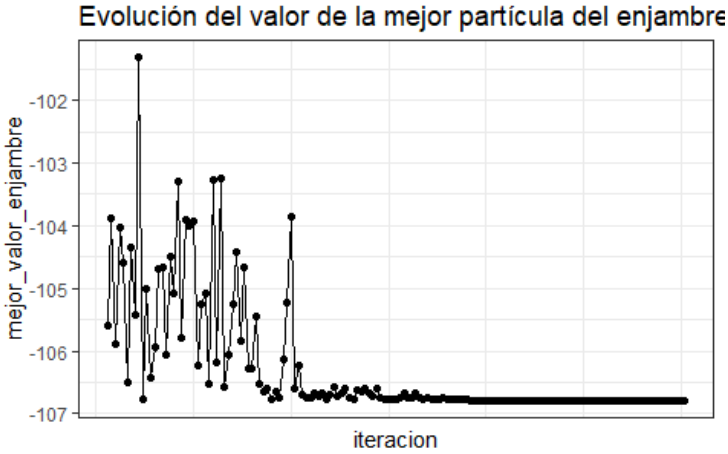
```
ggplot(data = datos, aes(x = x1, y = x2, z = f_x)) +
  geom_contour(aes(colour = stat(level)), bins = 20) +
  annotate(
    geom = "point",
    x = optimizacion$optim_posicion[1],
    y = optimizacion$optim_posicion[2],
    color = "red",
    size = 3
  ) +
  labs(title = "f(x1,x2) = x1^2 + x2^2") +
  theme_bw() +
  theme(legend.position = "none")
```



Evolución del enjambre

En el siguiente gráfico se puede ver cómo evoluciona el valor de la mejor partícula del enjambre a medida que avanzan las iteraciones. Se excluyen las dos primeras iteraciones para centrar la gráfica.

```
ggplot(data = optimizacion$resultados_df %>% filter(iteracion > 2),
       aes(x = iteracion, y = mejor_valor_enjambre)) +
  geom_line(aes(group = 1)) +
  geom_point() +
  labs(title = "Evolución del valor de la mejor partícula del enjambre") +
  theme_bw() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank())
```



Animación de cómo avanza la búsqueda del mínimo (para formato HTML)

```
library(gganimate)

# Se extrae la posición de cada partícula en cada iretación.
extraer_particulas <- function(enjambre){
                        particulas <- enjambre[["particulas"]]
                        particulas <- map(.x = particulas,
                                          .f = function(x){x[["posicion"]]})
                        return(particulas)
                      }

particulas <- enframe(optimizacion$historico_enjambre,
                      name = "iteracion",
                      value = "enjambre") %>%
              mutate(particulas = map(enjambre, .f = extraer_particulas)) %>%
  select(-enjambre) %>%
              unnest() %>%
              mutate(x1 = map_dbl(particulas, .f = function(x){x[1]}),
                     x2 = map_dbl(particulas, .f = function(x){x[2]})) %>%
```

```
              select(-particulas) %>%
              mutate(iteracion = stringr::str_remove(iteracion, "iter_"),
                     iteracion = as.numeric(iteracion))

gift <- ggplot() +
        geom_contour(data = datos,
                     aes(x = x1, y = x2, z = f_x, colour = stat(level)),
                     bins = 20) +
        geom_point(data = particulas,
                   aes(x = x1, y = x2),
                   color = "red",
                   size = 1.8) +
        theme_bw() +
        theme(legend.position = "none") +
        transition_manual(frames = iteracion) +
        ggtitle("Posición del mínimo encontrado",
                subtitle = "Generación {frame}")

animate(plot = gift, fps = 3)
```

Ejemplo 3

En este ejemplo se pretende evaluar la capacidad del algoritmo genético para encontrar el mínimo de la función de *Ackley*.

$$f(x_1, x_2) = -20exp[-0.2\sqrt{0.5(x_1^2 + x_2^2)}] - exp[0.5(\cos 2\pi x_1 + \cos 2\pi x_2)] + e + 20$$

la función tiene múltiples mínimos locales y un único el mínimo global que se encuentra en:

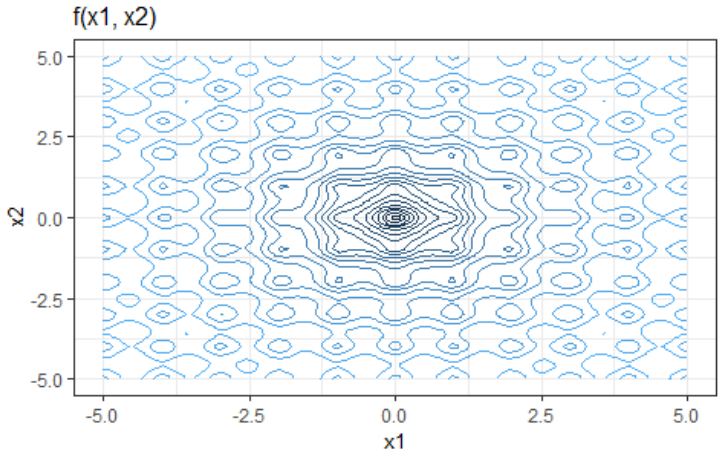
$$f(0,0) = 0$$

Función objetivo

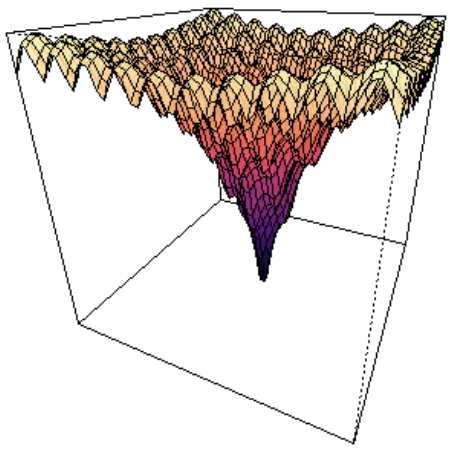
```
# Función objetivo a optimizar.
funcion <- function(x1, x2){
  -20*exp(-0.7*sqrt(0.5*(x1^2 + x2^2))) - exp(0.5*(cos(2*pi*x1) + cos(2*pi*x2))) +
exp(1) + 20
}
```

Representación gráfica de la función.

```
x1 <- seq(-5, 5, length.out = 100)
x2 <- seq(-5, 5, length.out = 100)

datos <- expand.grid(x1 = x1, x2 = x2)
datos <- datos %>%
         mutate(f_x = map2_dbl(x1, x2, .f = funcion))

ggplot(data = datos, aes(x = x1, y = x2, z = f_x)) +
  geom_contour(aes(colour = stat(level)), bins = 20) +
  labs(title = "f(x1, x2)") +
  theme_bw() +
  theme(legend.position = "none",
        title = element_text(size = 10))
```




```
x1  <- seq(-5, 5, length.out = 50)
x2  <- seq(-5, 5, length.out = 50)
f_x <- outer(x1, x2, FUN = funcion)

library(viridis)
colores        <- viridis::magma(n = 100, alpha = 0.9)
z.facet.center <- (f_x[-1, -1] + f_x[-1, -ncol(f_x)] +
                     f_x[-nrow(f_x), -1] +
                     f_x[-nrow(f_x), -ncol(f_x)])/4
z.facet.range  <- cut(z.facet.center, 100)

par(mai = c(0,0,0,0))
persp(x = x1, y = x2, z = f_x,
      shade = 0.8,
      r     = 1,
      phi   = 25,
      theta = 25,
      col   = colores[z.facet.range],
      axes = FALSE)
```



Optimización

```
optimizacion <- optimizar_enjambre(
                 funcion_objetivo = funcion,
                 n_variables      = 2,
                 optimizacion     = "minimizar",
                 limites_inf      = c(-5, -5),
                 limites_sup      = c(5, 5),
                 n_particulas     = 100,
                 n_iteraciones    = 500,
                 inercia          = 0.8,
                 reduc_inercia    = TRUE,
                 inercia_max      = 0.9,
                 inercia_min      = 0.4,
                 peso_cognitivo   = 2,
                 peso_social      = 2,
                 parada_temprana  = TRUE,
                 rondas_parada    = 5,
                 tolerancia_parada = 10^-8,
                 verbose          = FALSE
                )
```

```
## [1] "Algoritmo detenido en la iteración 317 por falta cambio absoluto mínimo de
1e-08 durante 5 iteraciones consecutivas."
```

Resultados

El objeto devuelto por la función `optimizar_enjambre()` almacena la información (valor, posición,...) de las partículas del enjambre en cada iteración.

```
head(optimizacion$resultados_df)
```

```
##     iteracion mejor_valor_enjambre
## 1           1         5.367426e+00
## 2           2         1.805455e-01
## 3           3         7.194989e-01
## 4           4         2.678177e+00
## 5           5         1.492813e+00
## 6           6         1.759362e+00
## 7           7         6.446600e-01
## 8           8         3.704404e+00
## 9           9         3.605608e-01
## 10         10         9.222856e-01
```

```
##                          mejor_posicion_enjambre diferencia_abs
## 1           0.100523943547159, 0.395375799853355             NA
## 2         0.00622189228871517, -0.016342169119997   5.186881e+00
## 3          0.0219475065748735, 0.0592269541040151   5.389534e-01
## 4         -0.0455211791278172, -0.194054800326038   1.958678e+00
## 5            0.00853298740417929, 0.1193356217513   1.185365e+00
## 6          -0.0358982695799994, 0.132956098356146   2.665496e-01
## 7         0.0570105621418058, 0.00543761623315797   1.114702e+00
## 8           -0.14973471150767, -0.219334880958335   3.059744e+00
## 9         -0.0171892004729894, 0.0289345989124111   3.343843e+00
## 10         0.0385986746231832, 0.0684982926428017   5.617248e-01
```

Mejor individuo

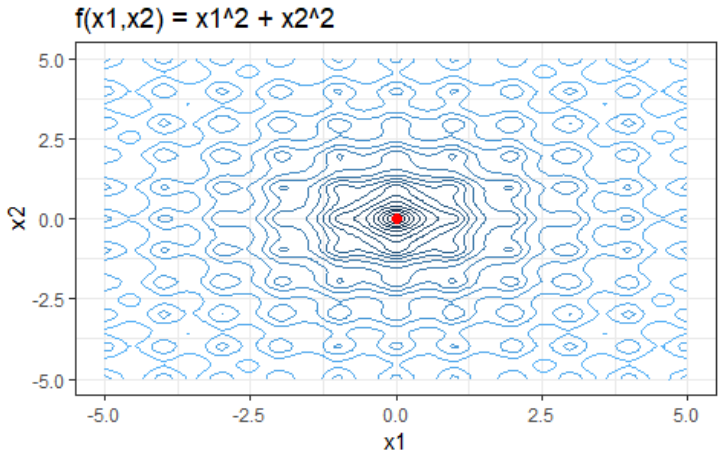
```
optimizacion$optim_posicion
```

```
## [1] 1.723748e-09 1.819586e-10
```

```
optimizacion$optim_valor
```

```
## [1] 1.715904e-08
```

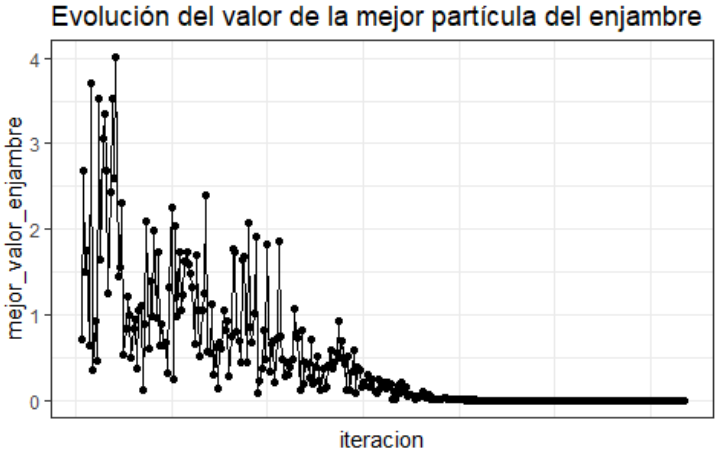
```
ggplot(data = datos, aes(x = x1, y = x2, z = f_x)) +
  geom_contour(aes(colour = stat(level)), bins = 20) +
  annotate(
    geom = "point",
    x = optimizacion$optim_posicion[1],
    y = optimizacion$optim_posicion[2],
    color = "red",
    size = 2
  ) +
  labs(title = "f(x1,x2) = x1^2 + x2^2") +
  theme_bw() +
  theme(legend.position = "none")
```



Evolución del enjambre

En el siguiente gráfico se puede ver cómo evoluciona el valor de la mejor partícula del enjambre a medida que avanzan las iteraciones. Se excluyen las dos primeras iteraciones para centrar la gráfica.

```
ggplot(data = optimizacion$resultados_df %>% filter(iteracion > 2),
       aes(x = iteracion, y = mejor_valor_enjambre)) +
  geom_line(aes(group = 1)) +
  geom_point() +
  labs(title = "Evolución del valor de la mejor partícula del enjambre") +
  theme_bw() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank())
```



Animación de cómo avanza la búsqueda del mínimo (para formato HTML)

```
library(gganimate)

# Se extrae la posición de cada partícula en cada iteración.
extraer_particulas <- function(enjambre){
                        particulas <- enjambre[["particulas"]]
                        particulas <- map(.x = particulas,
                                          .f = function(x){x[["posicion"]]})
                        return(particulas)
                      }

particulas <- enframe(optimizacion$historico_enjambre,
                      name = "iteracion",
                      value = "enjambre") %>%
              mutate(particulas = map(enjambre, .f = extraer_particulas)) %>%
              select(-enjambre) %>%
              unnest() %>%
              mutate(x1 = map_dbl(particulas, .f = function(x){x[1]}),
                     x2 = map_dbl(particulas, .f = function(x){x[2]})) %>%
```

```
              select(-particulas) %>%
              mutate(iteracion = stringr::str_remove(iteracion, "iter_"),
                     iteracion = as.numeric(iteracion))

gift <- ggplot() +
        geom_contour(data = datos,
                     aes(x = x1, y = x2, z = f_x, colour = stat(level)),
                     bins = 20) +
        geom_point(data = particulas,
                   aes(x = x1, y = x2),
                   color = "red",
                   size = 1.8) +
        theme_bw() +
        theme(legend.position = "none") +
        transition_manual(frames = iteracion) +
        ggtitle("Posición del mínimo encontrado",
                subtitle = "Generación {frame}")

animate(plot = gift, fps = 3)
```

Bibliografía

Particle Swarm Optimization: A TutorialJames BlondinSeptember 4, 2009

Sengupta, Saptarshi & Basak, Sanchita & Alan Peters II, Richard. (2018). Particle Swarm Optimization: A survey of historical and recent developments with hybridization perspectives. 10.3390/make1010010.

Bonyadi, Mohammad reza & Michalewicz, Zbigniew & Li, Xiaodong. (2014). An analysis of the velocity updating rule of the particle swarm optimization algorithm. Journal of Heuristics.

https://en.wikipedia.org/wiki/Particle_swarm_optimization