HTTPS://GITHUB.COM/AMORAC4/LIBRERIA-BIOINSPIRADOS.GIT

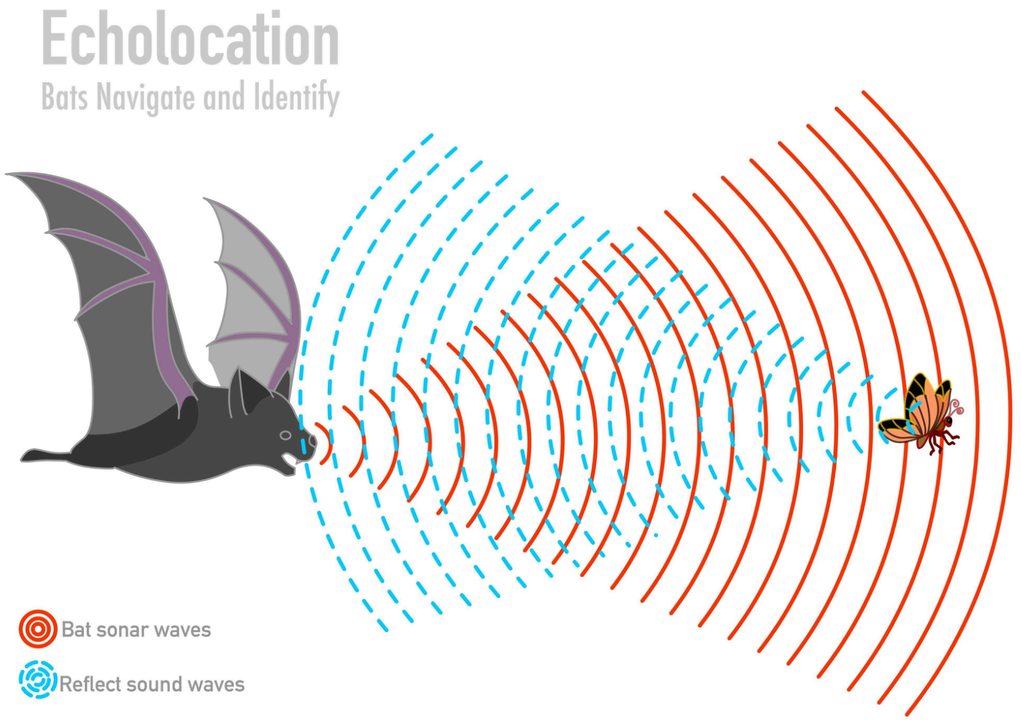
BAT ALGORITHM

Una Metaheurística Bio-inspirada para Optimización Continua

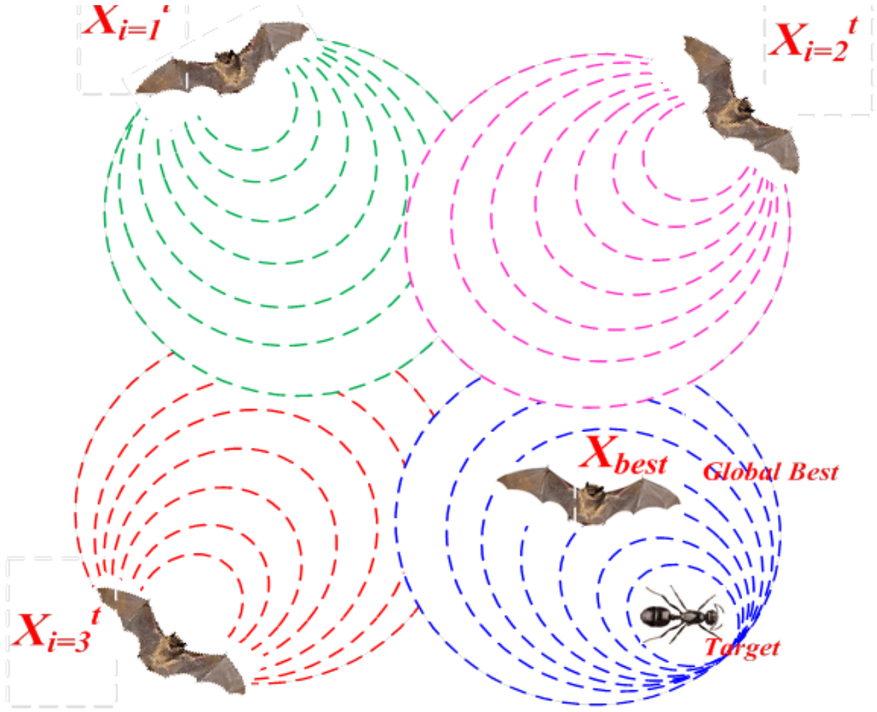
Adolfo Mora Córdova
Computo Evolutivo y Bioinspirado.

¿EN QUÉ SE INSPIRA?

El comportamiento de ecolocalización de los micro-murciélagos para cazar.



LA ANALOGÍA: CAZA EN LA OSCURIDAD



- **Murciélagos**: Una población de soluciones (X)
- **Espacio de Búsqueda:** El "cuarto oscuro" donde se busca el óptimo.
- **Presa:** La mejor solución encontrada (Mínimo Global, G).
- **Ecolocalización:** El mecanismo de búsqueda, que se divide en dos fases:
  - **Búsqueda Global (Exploración):** Volar lejos, gritos fuertes (A alta) y lentos (r baja).
  - **Búsqueda Local (Explotación):** Cerca de la presa, gritos silenciosos (A baja) y rápidos (r alta).

PARÁMETROS CLAVE DEL ALGORITMO

- **X**: Posición de los murciélagos (las soluciones).
- **V**: Velocidad de los murciélagos.
- **G**: Mejor solución global (la "presa").
- **Q**: Frecuencia (controla el tamaño del "salto" hacia G).
- **A**: Sonoridad (Loudness)
  * Controla la probabilidad de aceptar una nueva solución.
  * Inicia ALTA (ej. 0.95) y disminuye.
- **r**: Tasa de Pulso (Pulse Rate)
  * Controla la probabilidad de hacer búsqueda local.
  * Inicia BAJA (ej. 0.1) y aumenta.

```
rng = np.random.default_rng(seed)
dim = bounds.shape[0]

# Parámetros del algoritmo
A_vec = np.full(pop_size, A) # Vector de Sonoridad (Loudness)
R_vec = np.full(pop_size, r) # Vector de Tasa de Pulso (Pulse Rate)

# Inicializa las posiciones de los murciélagos
X = rand_vec_in_bounds_np(bounds, pop_size, rng)
# Inicializa las velocidades
V = np.zeros((pop_size, dim))

# Evalúa la población inicial
F = objective(X)

# Encuentra el mejor global inicial
g_idx = np.argmin(F)
```

PASO 1: BÚSQUEDA GLOBAL

```
# --- Generar nuevas soluciones (movimiento global) ---

# Genera frecuencias aleatorias (beta)
Q = Qmin + (Qmax - Qmin) * rng.random((pop_size, 1))

# Ecuación de velocidad
V = V + (X - G) * Q

# Ecuación de posición
X_new = X + V
```

Para cada murciélago en la población:

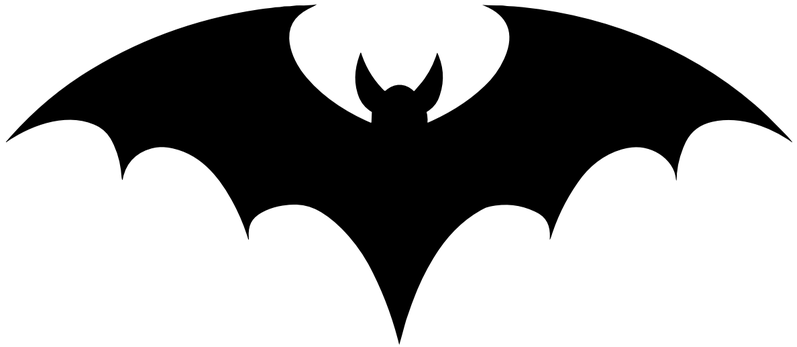
1. Generar Frecuencia (Q):
   * Se genera un Q aleatorio en el rango [Qmin, Qmax].
2. Actualizar Velocidad (V):
   * Se mueve hacia el mejor global G (similar a PSO).
   * V_new = V_old + (X_old - G) * Q
3. Calcular Posición Global (X_new):
   * X_new = X_old + V_new

PASO 2: BÚSQUEDA LOCAL

```
# --- Búsqueda local (paseo aleatorio) ---

# Identifica qué murciélagos harán búsqueda local (si rnd > r)
local_search_mask = rng.random(pop_size) > R_vec
num_local = np.sum(local_search_mask)

if num_local > 0:
    # Genera un paseo aleatorio alrededor de la *mejor* solución (G)
    # El 'sigma' controla qué tan lejos es el paseo
    avg_F = np.mean(F) # Promedio de fitness actual
    epsilon = sigma * avg_F if avg_F != 0 else sigma # Promedio de sonoridad

    walk = G + epsilon * rng.standard_normal(size=(num_local, dim))

    # Reemplaza las soluciones de X_new con el paseo aleatorio
    X_new[local_search_mask] = walk
```

4. Decidir Búsqueda Local:
   * Se genera un aleatorio rnd.
   * SI rnd > r (Tasa de Pulso):
     - El murciélago decide "olfatear" (explotar la zona).
     - Ignora la X_new calculada en el paso anterior.
     - Genera una nueva X_new dando un "pequeño salto" (paseo aleatorio) alrededor de la mejor solución G.
   -X_new = G + ε * rand_normal()
   -(En nuestro código: ε = sigma * avg(F))

PASO 3: ACEPTACIÓN Y ACTUALIZACIÓN



```python
# Evalúa las nuevas soluciones
F_new = objective(X_new)

# --- Aceptación de soluciones (Loudness y Pulso) ---

# Máscara de aceptación:
# 1. La nueva solución es mejor (F_new < F)
# 2. Y un número aleatorio es menor que la Sonoridad (A_vec)
accept_mask = (F_new < F) & (rng.random(pop_size) < A_vec)

# Actualiza las posiciones y fitness de los murciélagos aceptados
X[accept_mask] = X_new[accept_mask]
F[accept_mask] = F_new[accept_mask]

# Actualiza los parámetros A y r para los murciélagos aceptados
# A (Loudness) disminuye
A_vec[accept_mask] *= alpha
# R (Pulse rate) aumenta (se acerca a r_inicial)
R_vec[accept_mask] = r * (1 - np.exp(-gamma * (it + 1)))

# --- Actualizar el mejor global ---
# Revisa si alguna de las *nuevas* posiciones aceptadas es el
# nuevo mejor global.
g_idx = np.argmin(F)
if F[g_idx] < Gf:
    Gf = F[g_idx]
    G = X[g_idx].copy()
```

5. Evaluar X_new:
   * Se calcula el fitness de la nueva posición F(X_new).
6. Decidir Aceptación:
   * Se genera un aleatorio rnd_A.
   * SI F(X_new) < F(X_old) Y rnd_A < A (Sonoridad):
     - El murciélago acepta la nueva solución: X = X_new.
     - "Encontró la presa", por lo que actualiza sus parámetros:
       – Disminuye su Sonoridad: A = A * α (se vuelve "silencioso").
     – Aumenta su Tasa de Pulso: r = r_inicial * (1 - exp(-γ * t)) (grita "rápido").
7. Actualizar G:
   * Se revisa si X_new es la nueva mejor solución global.

CONCLUSIONES

- El Algoritmo de Murciélago (BA) es una metaheurística híbrida robusta.
- Combina exitosamente la inteligencia de enjambre (movimiento global) con la búsqueda local intensiva.
- Su característica única es el uso de la Sonoridad (A) y la Tasa de Pulso (r) como parámetros dinámicos que controlan la convergencia.
- Ha demostrado ser muy eficaz para problemas de optimización continua.

THANK YOU FOR YOUR ATTENTION