

# Algoritmo de PSO

Victor Gerardo Rodríguez Barragán

16 de Septiembre del 2023



## Descripcion

El algoritmo de PSO (Particle Swarm Optimization) es un algoritmo de optimizacion basado en poblaciones, en el cual se tiene un conjunto de particulas que se mueven en el espacio de busqueda y cada una de ellas tiene una velocidad y una posicion, la velocidad se actualiza en cada iteracion de acuerdo a la mejor posicion que ha encontrado la particula y la mejor posicion que ha encontrado el enjambre de particulas.

Nota: En el ejemplo al enjambre y a las particulas se les relaciona con un grupo de abejas. El ejemplo proporcionado por el profesor en clase fue util ya que fue un poco mas facil para mi entender el algoritmo como un enjambre de abejas que siguen una fuente de alimento (como las flores por ejemplo) y que cada abeja tiene una velocidad y una posicion la cual van actualizandose de manera gradual conforme van encontrando una mejor fuente.

## Codigo

```
/* El algoritmo fue sacado casi en su totalidad de:
https://github.com/SimSmith/PSO-rust
*/
extern crate rand;

use rand::Rng;

const N: usize = 40;
const ITERACIONES: u32 = 100_000;
const X_MAX: f64 = 5.0;
const X_MIN: f64 = -5.0;
const C1: f64 = 1.0;
const C2: f64 = 0.1;
const W: f64 = 1.5;
const A: f64 = 0.9;
const T: f64 = 1.0;

type Abejas = Vec<Abeja>;

#[derive(Debug)]
struct Abeja {
    posicion: (f64, f64),
    velocidad: (f64, f64),
    mejor_pos: (f64, f64),
}

pub fn pso() {
```

```

let mut abejas = iniciar(N, X_MAX, X_MIN, A, T);
let mut mejor_abeja_pos = (X_MIN, X_MAX);

for i in 0..ITERACIONES {
    let abejas_pos: Vec<f64> = abejas.iter().map(|x| evaluar(x.posicion)).collect();
    let abejas_mejor_pos: Vec<f64> = abejas.iter().map(|x| evaluar(x.mejor_pos)).collect();

    actualizar(&mut abejas, &mut mejor_abeja_pos, &abejas_pos, &abejas_mejor_pos, i as usize);

    if 1.0 / evaluar(mejor_abeja_pos) < 0.0001 {
        println!("Salio en iteracion {:?}", i);
        break;
    }
}
let mejor_abeja = evaluar(mejor_abeja_pos);
println!("Mejor abeja: {:?} con valor {:?}", mejor_abeja_pos, mejor_abeja);
}

fn iniciar(
    n: usize,
    x_max: f64,
    x_min: f64,
    a: f64,
    t: f64,
) -> Abejas {
    let mut particulas: Vec<Abeja> = Vec::with_capacity(n);
    let mut random = rand::thread_rng();

    for _ in 0..n {
        let r1: f64 = random.gen();
        let r2: f64 = random.gen();
        let r3: f64 = random.gen();
        let r4: f64 = random.gen();

        let x1 = x_min + r1 * (x_max - x_min);
        let x2 = x_min + r2 * (x_max - x_min);
        let v1 = a/t*(-(x_max-x_min)/2.0 + r3*(x_max-x_min));
        let v2 = a/t*(-(x_max-x_min)/2.0 + r4*(x_max-x_min));

        let particula = Abeja {
            posicion: (x1, x2),
            velocidad: (v1, v2),
            mejor_pos: (x1, x2),
        };

        particulas.push(particula);
    }
}

```

```

    }

    // println!("{:?}", particulas);
    particulas
}

fn evaluar((x, y): (f64, f64)) -> f64 {
    1.0 / (x*x + y - 11.0).powi(2) + (x + y*y - 7.0).powi(2)
}

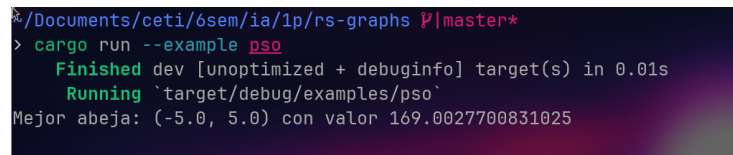
fn actualizar(
    particulas: &mut Abejas,
    mejor_abeja_pos: &mut (f64, f64),
    x_particula: &Vec<f64>,
    x_mejor_particula: & Vec<f64>,
    iter: usize,
) {
    let mut mejor_abeja = evaluar(*mejor_abeja_pos);

    for i in 0..particulas.len() {
        if x_particula[i] > x_mejor_particula[i] {
            particulas[i].mejor_pos = particulas[i].posicion;

            if x_particula[i] > mejor_abeja {
                *mejor_abeja_pos = particulas[i].posicion;
                mejor_abeja = evaluar(*mejor_abeja_pos);
                println!("Mejor abeja: {:?} en iteracion {:?}", mejor_abeja_pos, iter);
            }
        }
    }
}
}

```

## Resultados



```

~/Documents/ceti/6sem/ia/1p/rs-graphs P|master*
> cargo run --example pso
    Finished dev [unoptimized + debuginfo] target(s) in 0.01s
    Running `target/debug/examples/pso`
Mejor abeja: (-5.0, 5.0) con valor 169.0027700831025

```

Cantidad de iteraciones: 100,000 (definido en el código)

## Conclusion

El algoritmo de PSO es un algoritmo de optimizacion basado en poblaciones, viendolo asi es muy similar a un algoitmo genetico, pero en este caso las particulas no se reproducen, sino que se mueven en el espacio de busqueda, en el cual cada una de ellas tiene una velocidad y una posicion, un poco complejo de entender pero muy interesante, lo veo util para encontrar la solucion a un problema de optimizacion en un espacio de busqueda desconocido o muy grande.

La implementacion es un poco compleja, revise pseudocodigos y ejemplos de implementaciones en varias paginas web y no me termino quedando claro. El codigo que implemente fue sacado de un repositorio y lo intente adaptar a mi problema, no quedo muy optimizado ni 100% funcional, seguire trabajando con el para ver si puedo mejorarlo.