Proyecto PSO

Licenciatura en Ciencia de Datos

Asignatura:
Computación concurrente

Maestro:

Oscar Alejandro Esquivel Flores

Alumnos:

David Ávila García
Santiago Licea Becerril

Introducción

La Optimización por Enjambres de Partículas (conocida como PSO, por sus siglas en inglés, Particle Swarm Optimization) es una técnica de optimización/búsqueda.

Este método fue descrito alrededor de 1995 por James Kennedy y Russell C. Eberhart (Kennedy, J. & Eberhart, R. (1995), 'Particle swarm optimization', Neural Networks, 1995. Proceedings., IEEE International Conference), y se inspira en el comportamiento de los enjambres de insectos en la naturaleza. En concreto, podemos pensar en un enjambre de abejas, ya que éstas a la hora de buscar polen buscan la región del espacio en la que existe más densidad de flores, porque la probabilidad de que haya polen es mayor. La misma idea fue trasladada al campo de la computación en forma de algoritmo y se emplea en la actualidad en la optimización de distintos tipos de sistemas

Ya comprendida la intuición del PSO, plantemos el modelo matemáticamente y el pseudocódigo del algoritmo. Son tres las principales ecuaciones con las que podemos describir el modelo PSO, la primera de ellas se refiere el proceso de inicialización de las posiciones de las partículas del enjambre,

$$X_k^{i,t} = l_k + rand(u_k - l_k),\ X_k^{l,t} \epsilon X^t$$

donde $X_k^{i,t}$ es la i-ésima partícula de la población X^t, i es el índice que se refiere al número de partículas y tiene como valor máximo el tamaño de la población. La dimensión del problema se define por la variable k y el número de iteraciones con la variable t. Mientras l_k y u_k son los límites inferior y superior respectivamente para una de las dimensiones del espacio de búsqueda, por último rand es un número aleatorio uniformemente distribuido entre el rango de cero a uno.

Para poder obtener la nueva posición que tendrán las partículas, primero es necesario calcular las velocidades de cada una.Inicialmente las partículas inician con velocidad cero, pero para la segunda iteración deben de calcularse las nuevas velocidades. En la siguiente ecuación se presenta como se lleva a cabo el cálculo de dicha velocidad.

$$V^{t+1} = V^t + rand1\ x\ (P - X^t) + rand2\ x\ (G - X^t)$$

Donde V^{t+1} es el valor de la velocidad que se calcula para la iteración t+1, V^t es la velocidad en la iteración anterior t, X^t es el vector que contiene las posiciones de cada partícula, P contiene las mejores posiciones actuales asociadas a la vecindad de cada partícula, mientas G es la mejor partícula a nivel global.Por otra parte, rand1 y rand2 son números aleatorios usualmente distribuidos de forma uniforme en el rango cero a uno.Las operaciones matemáticas se realizan de forma vectorial respetando las reglas del álgebra de vectores.

Después de calcular la velocidad, las partículas son desplazadas hacia nuevas posiciones en la iteración actual. Para realizar este movimiento se realiza una simple operación donde se combina la velocidad con las posiciones anteriores de la población, este operador se describe como:

$$X^{t+1} = X^t + V^{t+1}$$

Donde X^{t+1} Es el vector donde son almacenadas las nuevas posiciones obtenidas en la iteración t+1. X^t corresponde a las posiciones previas de las partículas. Finalmente V^{t+1} es el vector de velocidad que se obtuvo como anteriormente se mencionó

Pseudocódigo del PSO

Initialize population
for $t = 1$: maximum generation
  for $i = 1$: population size
    if $f\left(x_{i,d}(t)\right) < f\left(p_i(t)\right)$ **then** $p_i(t) = x_{i,d}(t)$
      $f\left(p_g(t)\right) = \min_i\left(f\left(p_i(t)\right)\right)$
    end
    for $d = 1$: dimension
      $v_{i,d}(t+1) = wv_{i,d}(t) + c_1 r_1\left(p_i - x_{i,d}(t)\right) + c_2 r_2\left(p_g - x_{i,d}(t)\right)$
      $x_{i,d}(t+1) = x_{i,d}(t) + v_{i,d}(t+1)$
      if $v_{i,d}(t+1) > v_{\max}$ **then** $v_{i,d}(t+1) = v_{\max}$
      else if $v_{i,d}(t+1) < v_{\min}$ **then** $v_{i,d}(t+1) = v_{\min}$
      end
      if $x_{i,d}(t+1) > x_{\max}$ **then** $x_{i,d}(t+1) = x_{\max}$
      else if $x_{i,d}(t+1) < x_{\min}$ **then** $x_{i,d}(t+1) = x_{\min}$
      end
    end
  end
end

Pruebas

Se modificó el código del PSO para que admitiera las funciones que se mencionan a continuación. También se graficaron las funciones para tener una visualización mejor y utilizando la biblioteca BenchmarkTools obtuvimos la media y la desviación estándar del tiempo de ejecución de cada prueba, evaluando las siempre con los mismos parámetros.

```
function PSO1(d, l, u, Part_N, Max_iter,f)

  #Proceso de inicialización de PSO
  x = l' .+ rand(Uniform(0,1), Part_N, d) .* (u - l)'
  #Evalua la función objetivo
  obj_func = [f(x[i]) for i=1:Part_N]
  #Obtiene el mejor valor global(mínimo)
  glob_opt = minimum(obj_func)
  ind = argmin(obj_func)
  #Vector de valores optimos
  G_opt = reshape(x[ind, :], 1, d) * ones(d, Part_N)
  Mejor_pos = x[ind, :]
  #Mejor local para cada partícula
  Loc_opt = x

  #Inicializa velocidades
  v = zeros(Part_N, d)

  #Vector para poder evaluar las nuevas posiciones
  #en la función objetivo
  Nva_obj_func = zeros(1, Part_N)
  Evol_func_obj = zeros(1, Max_iter)

  #Inicia proceso de optimización PSO
  t = 1
  #Criterio de paro
  while t < Max_iter
    #Calcula la nueva velocidad
    v = v .+ rand(Uniform(0,1), Part_N, d) .* (Loc_opt - x) .+ rand(d)' .* (G_opt' .- x)
    #Calcula nueva posición
    x = x .+ v
    #Se verifica que las partículas no se salan de los límites
    for i=1:Part_N
      if x[i, :] > u
        x[i, :] = u
      elseif x[i, :] < l
        x[i, :] = l
      end
      end
      #Se evalúan las nuevas posiciones en la funcion objetivo
      Nva_obj_func[i] = f(x[i])
      #Se verifica si se actualizaron los óptimos locales
      if Nva_obj_func[i] < obj_func[i]
        #Actualiza óptimo local
        Loc_opt[i, :] = x[i, :]
        #Actualiza función objetivo
        obj_func[i] = Nva_obj_func[i]
      end
    end
    #Obtiene el mejor valor global
    Nvo_glob_opt = minimum(obj_func)
    ind = argmin(obj_func)
    #Se verifica si se actualiza el óptimo global
    if Nvo_glob_opt < glob_opt
      glob_opt = Nvo_glob_opt
      #Obtenemos los valores óptimos en cada dimensión
      G_opt[:] = reshape(x[ind, :], 1, d) * ones(d, Part_N)
      Mejor_pos = x[ind, :]
    end
    #Almacena los valores de función objetivo en cada iteración
    Evol_func_obj[t] = glob_opt
    #Incrementa iteraciones
    t = t + 1
  end

  return Mejor_pos
end
```

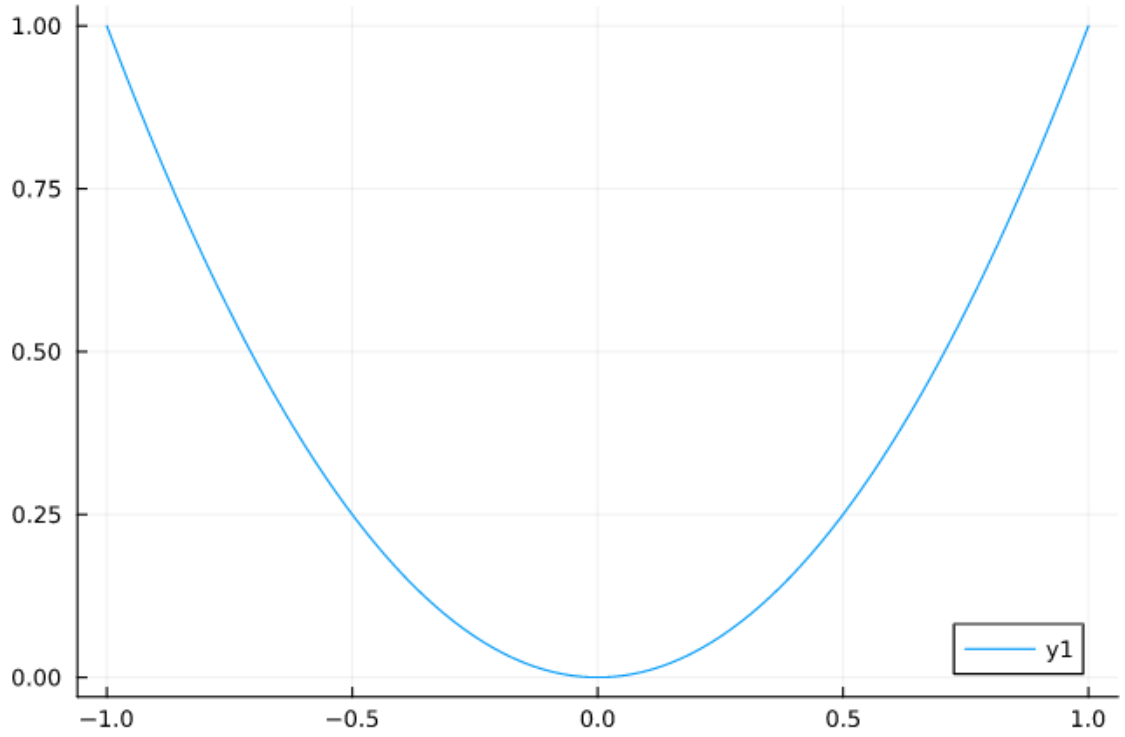
```
#Pruebas
#Procederemos a hacer 2 pruebas con 2 funciones diferentes
```

```
using Plots
```

```
function cuad(x)
    return x^2
end
```

```
cuad (generic function with 1 method)
```

```
#Gráfica de x², vemos que hay un mínimo en 0
x = range(-1,1, length=100)
plot(x,cuad)
```



```
PSO1(1,[-1],[1],100,100,cuad)
```

```
1-element Vector{Float64}:
 2.0788861496090494e-5
```

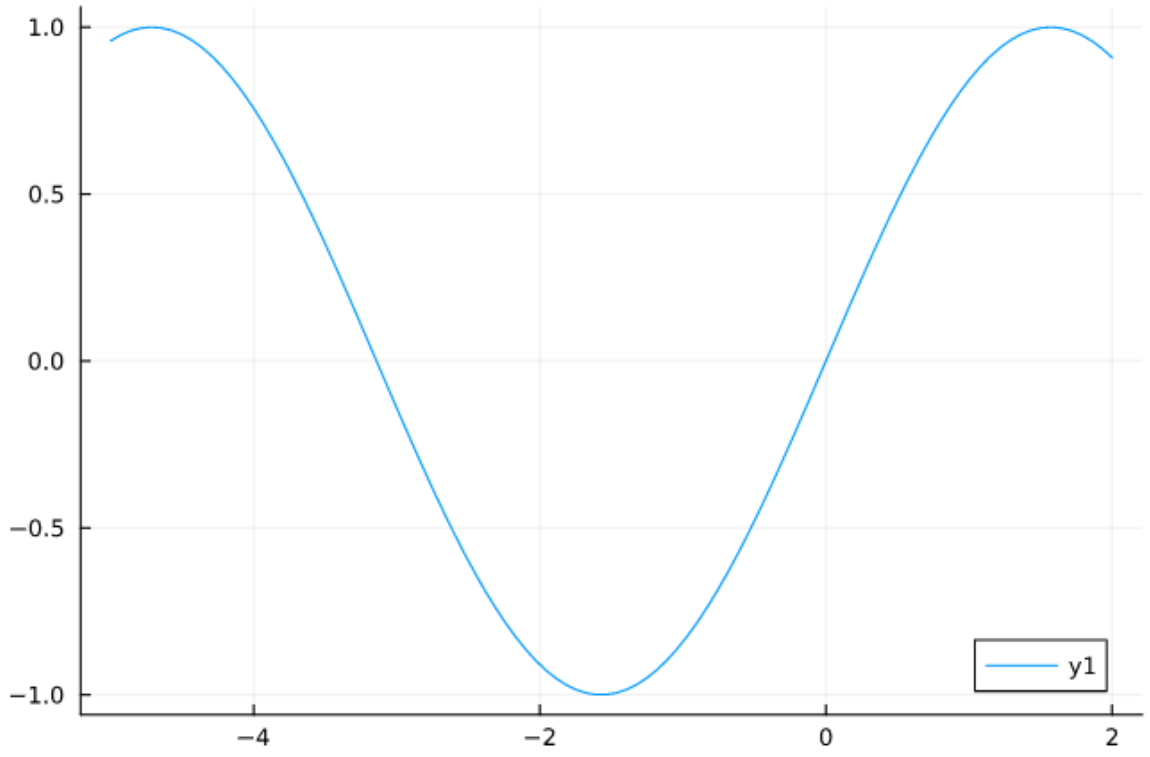
```
@benchmark PSO1(1,[-1],[1],100,100,cuad)
```

```
BenchmarkTools.Trial: 626 samples with 1 evaluation.
 Range (min … max):  7.317 ms …  13.546 ms  ┊ GC (min … max): 0.00% … 27.14%
 Time  (median):     7.738 ms               ┊ GC (median):    0.00%
 Time  (mean ± σ):   7.987 ms ± 827.966 µs  ┊ GC (mean ± σ):  1.63% ±  5.72%

  7.32 ms          Histogram: frequency by time         11.9 ms <

 Memory estimate: 2.51 MiB, allocs estimate: 81920.
```

```
function sen(x)
    return sin(x)
end
```

```
sen (generic function with 1 method)
```

```
#Gráfica de seno(x), donde vemos que tiene un mínimo en -pi/2
x = range(-5,2, length=100)
plot(x,sen)
```

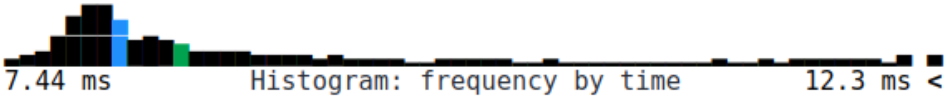



```
PSO1(1,[-3],[-1],100,100,sen)
```

```
1-element Vector{Float64}:
 -1.5709465910479046
```

```
@benchmark PSO1(1,[-3],[-1],100,100,sen)
```

```
BenchmarkTools.Trial: 602 samples with 1 evaluation.
 Range (min … max):  7.444 ms …  13.862 ms  ┊ GC (min … max): 0.00% … 28.06%
 Time  (median):     8.024 ms               ┊ GC (median):    0.00%
 Time  (mean ± σ):   8.305 ms ± 899.615 μs  ┊ GC (mean ± σ):  1.66% ±  5.81%
```




```
 Memory estimate: 2.46 MiB, allocs estimate: 80921.
```