Asignatura: Optimización
Profesor: D. Sc. Gerardo García Gil
2023-A
Ingeniería en Desarrollo de Software
Centro de Enseñanza Técnica Industrial (CETI)

Investigación.

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

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:

Crear un enjambre inicial de n

partículas aleatorias. 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.

Evaluar cada partícula con la función objetivo.

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.

Si no se cumple un criterio de parada, volver al paso 2.

Desarrollo.

1. Se hizo uso del código proporcionado por el profesor en clase, el siguiente script desarrolla el método PSO de aplicación. El script desarrolla lo siguiente:
2. Se inicializan las variables D, C, S, P y M con valores dados por el enunciado del problema.
3. Se define la función objetivo del problema utilizando la sintaxis de MATLAB. La función toma como argumento un vector xi, y se utiliza para evaluar el desempeño de una solución en el problema.
4. Se definen las variables N, d, lb y ub, que corresponden al número de partículas en el algoritmo PSO, las dimensiones de la búsqueda, los límites inferior y superior del espacio de búsqueda, respectivamente.
5. Se inicializan los valores de k, kmax, c1 y c2, que corresponden al número de iteraciones que se realizarán, al número máximo de iteraciones, y a las constantes de cognición y social del algoritmo PSO, respectivamente.
6. Se inicializan los vectores de posición y velocidad de las partículas utilizando valores aleatorios dentro del espacio de búsqueda.
7. Se evalúa la función objetivo para cada una de las partículas y se registran las posiciones y desempeños de las mejores partículas locales y la mejor partícula global.
8. Se inicia un ciclo while que se repetirá hasta que se alcance el número máximo de iteraciones o se cumpla un criterio de parada diferente.
9. Dentro del ciclo while se dibuja la superficie de optimización en una figura y se dibujan las posiciones de las partículas, utilizando diferentes colores para las mejores partículas locales y la mejor partícula global.
10. Se actualizan las velocidades de las partículas utilizando las ecuaciones del algoritmo PSO, que involucran las posiciones actuales de las partículas, las mejores posiciones locales y la mejor posición global encontrada hasta el momento.
11. Se actualizan las posiciones de las partículas utilizando las velocidades nuevas calculadas en el paso anterior. Además, se verifica que las posiciones de las partículas estén dentro del espacio de búsqueda.
12. Se evalúa la función objetivo para cada una de las partículas y se registran las posiciones y desempeños de las mejores partículas locales y la mejor partícula global encontrada hasta el momento.
13. Se actualiza la mejor partícula global y las mejores partículas locales si se encuentra una solución mejor que la actual.
14. Se registra la mejor solución encontrada en cada iteración del algoritmo.
15. Se dibuja una figura con la evolución de la mejor solución encontrada en cada iteración.
16. Se muestran por pantalla los resultados finales del algoritmo, que corresponden a la mejor solución encontrada, el desempeño de esta solución y el valor de la función objetivo correspondiente.

Código:

```
clear all %clear memory
close all %close matlab windows
D = 5000; %Annual demand
C = 5;    %Cost per unit
S = 49;   %Cost for order
P = 0.2;  %Percent storage cost
M = P*C   %Storage cost
funObj = @(xi) D*C + D/xi*S + xi/2*M; %Objective function
N = 5; %Particle number
d = 1; % Dimensions
lb = [400]; %Lower limit of search space
ub = [1200]; %Upper limit of search space
k = 0; %iteration
kmax = 150; %Maximun number of iterations
c1 = 2; %Cognitive constant
c2 = 2; %Social constant
%Initialization of particles and velocity
for i=1:N
   x(i,:) = rand(1,d).*(ub-lb)+lb; %initialization of particles
   v(i,:) = zeros(1,d);            % Velocity initialization
end
%%Evaluation of the initial particles with the objective function
for i=1:N
   xi=x(i,:)   %Extraction of the particle xi
   fx(i,:) = funObj(xi);  %Evaluation of the particle xi
end
%%Record of the best global particle and the best local particles
[gfit, ind] = min(fx); %Fitness of the best global particle
g = x(ind,:); %Location of the best global particle
```

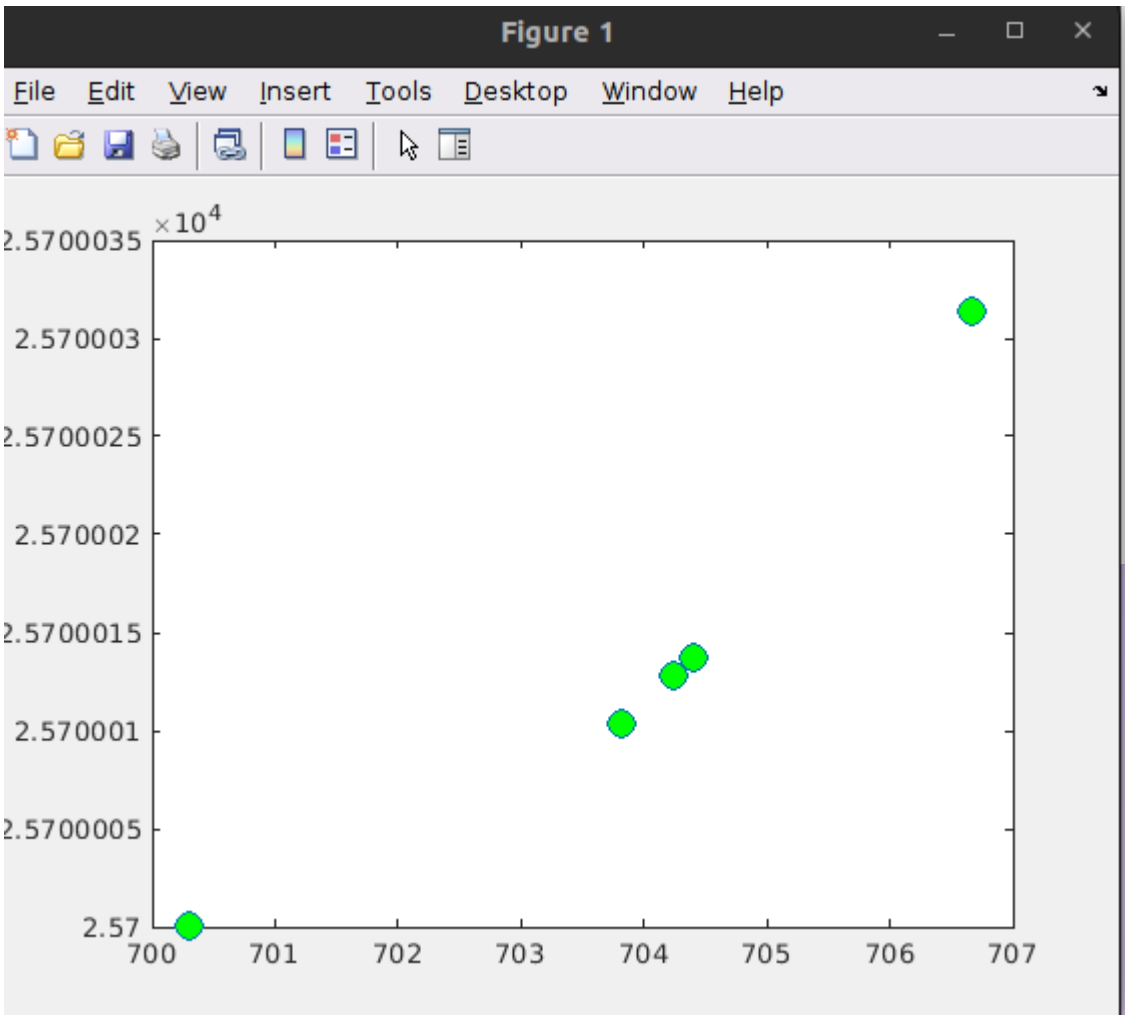
```
fp = fx;      %Fitness of the best local particle
p = x;        %Position of the best local particle
axisx = lb:ub; %solution vector
axisy = [];
for i = 1:length(axisx)
   axisy(i) = funObj(axisx(i));
end
%Iterative process
while k < kmax %Stop criterion
k = k+1; %New generation
%%The optimization surface is drawn
figure(1);
%Particles are drawn in red color
plot(x,fx,'o', 'MarkerFaceColor', 'm', 'MarkerSize', 10)
pause(0.3)
%Draw the best local particles in green
plot(p,fp,'o', 'MarkerFaceColor', 'g', 'MarkerSize', 10)
%Pause to allow visualization
pause(0.3)
hold off
%Compute the new velocity for each particle
for i=1:N
   xi = x(i,:); %Extraction of particle xi
   pi = p(i,:); %Extraction of local particle pi
    v(i,:) = v(i,:)+c1*rand(1,d).*(pi-xi)+c2*rand(1,d).*(g-xi); %Determination of the
new velocitity for each particle vi
end
%%Determination of the new position of each particle
x=x+v
for i=1:N
   for j=1:d
```

```
        if x(i,j) < lb(j)
            x(i,j) = lb(j);
        elseif x(i,j) > ub(j)
            x(i,j) = ub(j);
        end
    end
end
%%Evaluation of the new particle withc the objective function
for i=1:N
    xi = x(i,:);
    fx(i,:) = funObj(xi);
end
%Record of the best global particle and the best local particles
[gfitkplus1,ind] = min(fx);
%if a better solution is found, update the global particle
%fitness = desempeño
if gfitkplus1 < gfit
    %Update the fitness of the best global particle
    gfit = gfitkplus1;
    %Update the position of the best global particle
    g = x(ind,:);
end
for i=1:N
    %update your best local particle
    if fx(i,:)<fp(i,:)
        %Update the fitness of the best local particles
        fp(i,:) = fx(i,:);
        %Update the position on the best local particles
        p(i,:) = x(i,:);
    end
```

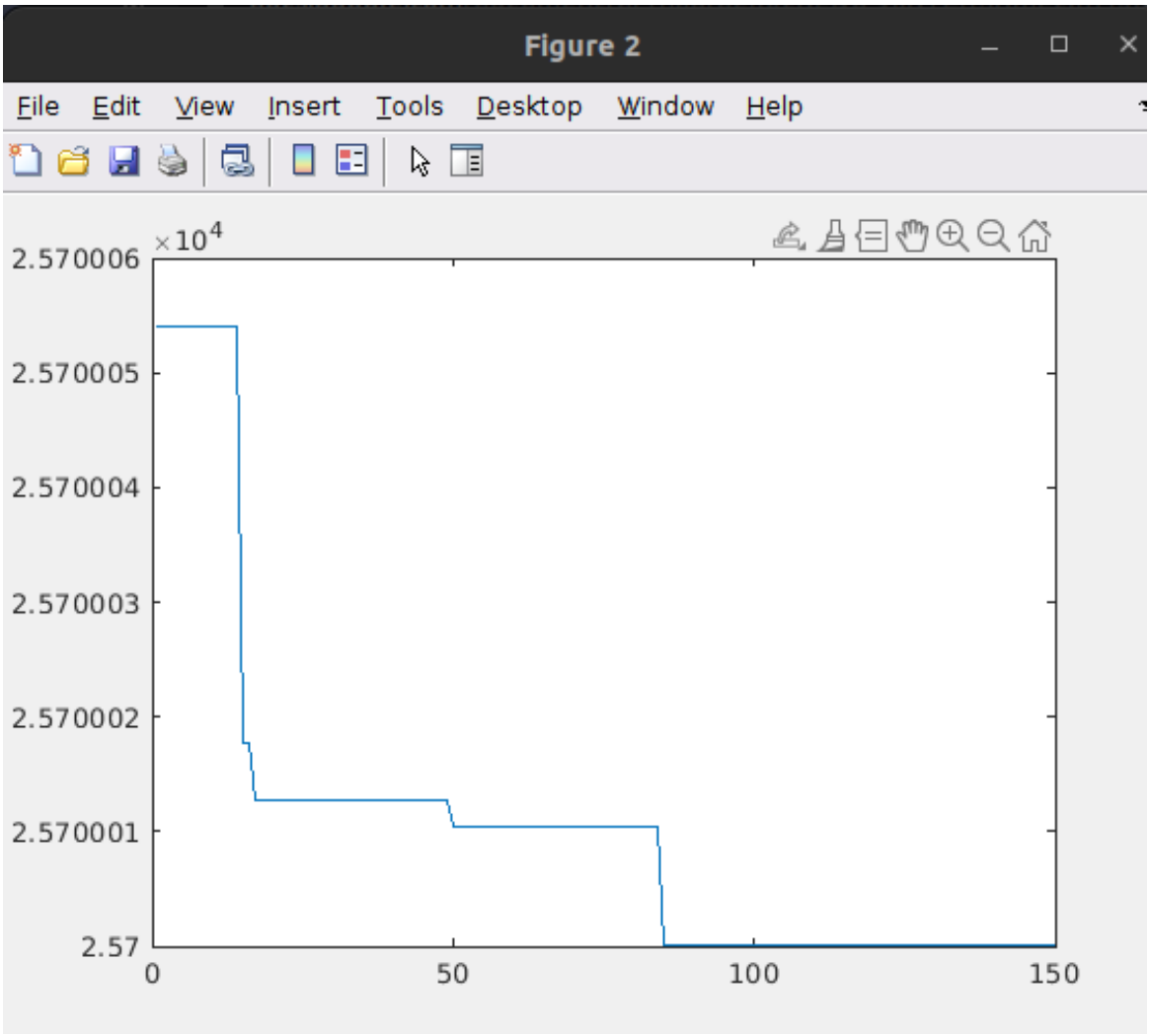
```
end
%Register the best solutions found in each generation
Evolution(k) = gfit;
end
figure
plot(Evolution)
disp(['Best Result:',num2str(g)])
disp(['Best Fitness:',num2str(g)])
disp(['Best:',num2str(gfit)]);
```

Resultados de la práctica.

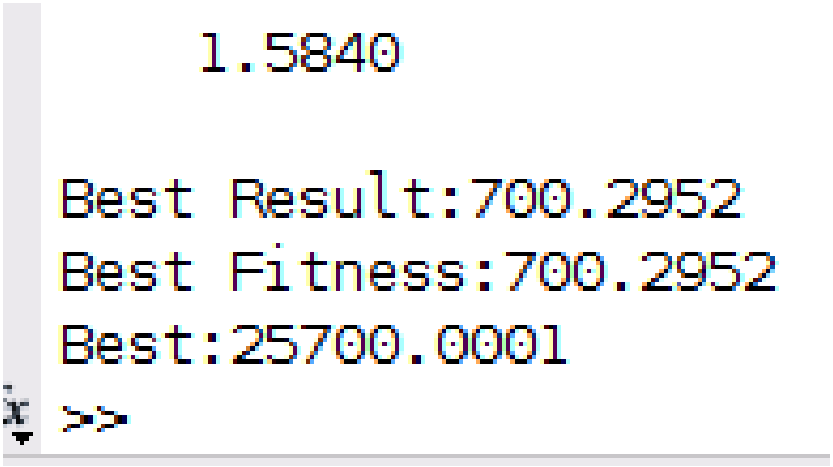
(vista en segunda dimensión con las partículas)



Gráfica de evolución del mejor ajuste a medida que avanza el algoritmo.



Resultado del consola con el dato obtenido.



Conclusiones.

Este código implementa el algoritmo PSO (Particle Swarm Optimization) para encontrar la solución óptima de un problema de inventario, maximizando la utilidad y minimizando los costos de almacenamiento y pedido.

Fue sencillo realizar esta práctica ya que contamos con el código, el trabajo de esta práctica es entender la práctica del algoritmo y poder replicarlo de ser necesario.

Referencias.

1. Caparrini, F. S. (s. f.). PSO: Optimización por enjambres de partículas - Fernando Sancho Caparrini. http://www.cs.us.es/~fsancho/?e=70

2. Kexugit. (2015, 14 agosto). Inteligencia artificial - optimización del enjambre de partículas. Microsoft Learn. https://learn.microsoft.com/es-es/archive/msdn-magazine/2011/august/artificial-intelligence-particle-swarm-optimization