# UNIVERSIDAD DE GUANAJUATO

DIVISIÓN DE INGENIERÍAS

CAMPUS – IRAPUATO SALAMANCA

Inteligencia Artificial

PROF. GARCÍA CAPULÍN CARLOS HUGO

TAREA 4

PSO: Funciones Crear, Inicializar, Mostrar, Evaluar

*Martínez Lona Verónica Montserrat*

*Lunes 27 Febrero, 2017*

## Introducción:

Se empezó a desarrollar el método POS, lo que se hizo fue empezar la implementación de las partículas y el enjambre. Se implementaron las funciones crear, inicializar, mostrar y evaluar.

## Desarrollo:

Primeramente se definió la estructura de una partícula, con sus componentes principales:

- Posición actual
- Velocidad actual
- Mejor posición
- Peso de la partícula

Después de se definió el enjambre, que básicamente es una colección de partículas. El enjambre se compone de:

- El número de patículas del enjambre
- El número de parámetros de las partículas
- Un arreglo de partículas

Los métodos que se implementaron en esta primera etapa del desarrollo del algoritmo POS fueron enfocados a la creación correcta del enjambre:

- *SWARM *CreateSwarm(const unsigned int numParticles, const unsigned int numParameters);*
  - Crea el enjambre: la colección de partículas (por lo que las partículas se crean dentro de la función), y asigna el número de parámetros y el número de partículas.
- *void InitSwarm(SWARM *pSwarm, const float LInf, const float Lsup);*
  - Inicializamos el enjambre, en esta etapa se hizo con datos al azar.
- *void ShowParticle(SWARM *pSwarm, const unsigned int i);*
  - Muestra la información contenida en la partícula.
- *void ShowSwarm(SWARM *pSwarm);*
  - Muestra todas las partículas que contiene el enjambre.
- *void DeleteSwarm(SWARM *pSwarm);*
  - Liberamos la memoria utilizada por el enjambre.
- *void EvaluateSwarm(SWARM *pSwarm);*
  - Evalúa cada partícula individualmente en la ecuación de prueba.

# Código:

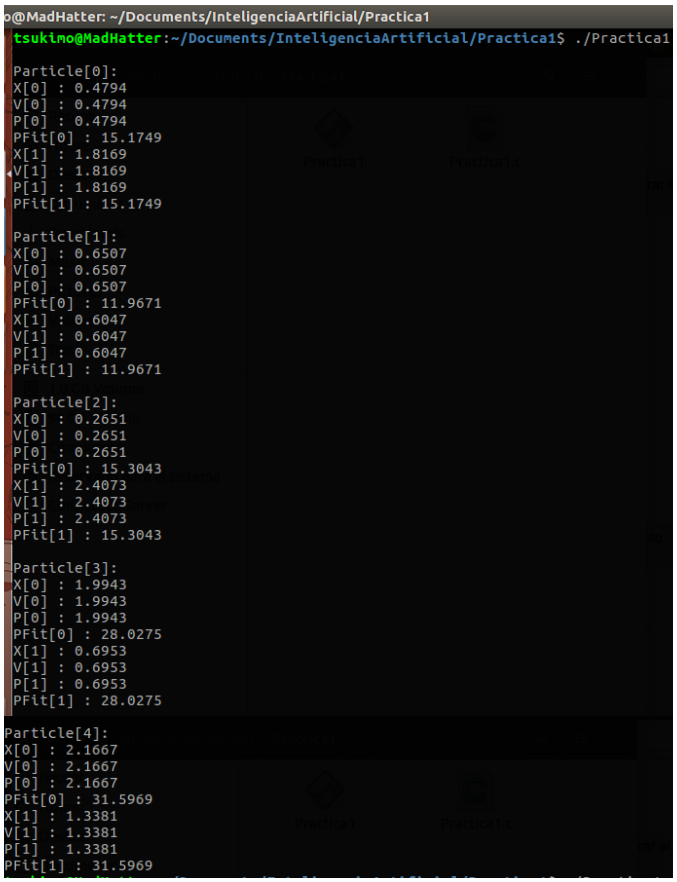
```
//Desarrollo de PSO
/*
Problema de ejemplo:
  Maximizar la función:
    f(x, y) = 50 - (x-5) ^ 2 + (y - 5) ^ 2
*/
#include<stdio.h>
#include<stdlib.h>

//Particle definition
typedef struct {
  float *Xi; //Actual Position
  float *Vi; //Actual Vel
  float *Pi; //Best positiom
  float XFit; //Resultado directo de la ecuación
} PARTICLE;

//Swarm definition
typedef struct {
  unsigned int numParticles;
  unsigned int numParameters;
  PARTICLE *particle; //swarm
} SWARM;

const unsigned int NumberParticles = 5;
const unsigned int NumberParameters = 2;
const float LimiteInf = 0;
const float LimiteSup = 10;

SWARM *CreateSwarm(const unsigned int numParticles, const unsigned int numParameters);
void InitSwarm(SWARM *pSwarm, const float LInf, const float LSup);
void ShowParticle(SWARM *pSwarm, const unsigned int i);
void ShowSwarm(SWARM *pSwarm);
void DeleteSwarm(SWARM *pSwarm);
void EvaluateSwarm(SWARM *pSwarm);

int main(void)
{
  SWARM *expSwarm;

  expSwarm = CreateSwarm(NumberParticles, NumberParameters);
  InitSwarm(expSwarm, LimiteInf, LimiteSup);
  EvaluateSwarm(expSwarm);
  ShowSwarm(expSwarm);
  DeleteSwarm(expSwarm);

  return 0;
}

SWARM *CreateSwarm(const unsigned int numParticles, const unsigned int numParameters)
{
  SWARM *pSwarm;

  //_____________________________________Asignar memoria para la estructura enjambre
  pSwarm = (SWARM*)malloc(sizeof(SWARM));
  if (pSwarm == NULL)
  {
    printf("Error al asignar memoria a la estructura enjambre.\n");
    return -1;
```

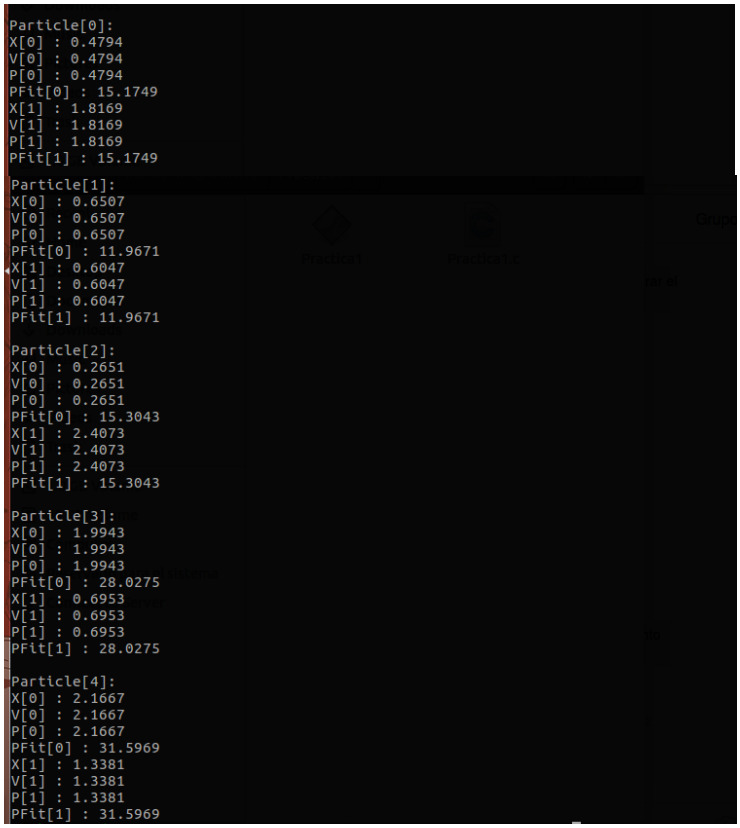
```
 }
 //_____Inicializar la estructura
 //Asignar número de partículas
 pSwarm -> numParticles = numParticles;
 //Asignar memoria para las partículas
 pSwarm -> particle = (PARTICLE*)malloc(numParticles*sizeof(PARTICLE));
 if(pSwarm -> particle == NULL)
 {
  printf("Error al asignar memoria a las particulas.\n");
  return -1;
 }
 pSwarm -> numParameters = numParameters;
 //Asignar la memoria a cada partícula
 for (unsigned int i = 0; i < numParticles; i++)
 {
  pSwarm -> particle[i].Xi = (float*)malloc(numParameters*sizeof(float));
  pSwarm -> particle[i].Vi = (float*)malloc(numParameters*sizeof(float));
  pSwarm -> particle[i].Pi = (float*)malloc(numParameters*sizeof(float));
 }

 return pSwarm;
}

void InitSwarm(SWARM *pSwarm, const float infL, const float supL)
{
 unsigned int i, j;
 double r;
 //Para todas las partículas
 for(i = 0; i < pSwarm -> numParticles; i++)
 {
  //Para todos los parámetros
  for(j = 0; j < pSwarm -> numParameters; j++)
  {
   r = ((double) rand() / RAND_MAX) * (supL - infL) + infL; //Valor random entre InfL-SupL
   pSwarm -> particle[i].Xi[j] = r;
   pSwarm -> particle[i].Vi[j] = r;
   pSwarm -> particle[i].Pi[j] = r;
  }
 }
 return 0;
}

void EvaluateSwarm(SWARM *pSwarm)
{
 unsigned int i;
 for(i = 0; i < pSwarm -> numParticles; i++)
 {
  pSwarm -> particle[i].XFit = 50 - ((pSwarm -> particle[i].Xi[0] - 5) * (pSwarm -> particle[i].Xi[0] - 5) +
                (pSwarm -> particle[i].Xi[1] -5 ) * (pSwarm -> particle[i].Xi[0] - 5));
 }
 return 0;
}

void ShowParticle(SWARM *pSwarm, const unsigned int i)
{
 unsigned int j;

 printf("\nParticle[%d]: \n", i);

 for(j = 0; j < pSwarm -> numParameters; j++)
 {
  printf("X[%d] : ", j);
```

```
    printf("%2.4f \n", pSwarm -> particle[i].Xi[j]);
    printf("V[%d] : ", j);
    printf("%2.4f \n", pSwarm -> particle[i].Vi[j]);
    printf("P[%d] : ", j);
    printf("%2.4f \n", pSwarm -> particle[i].Pi[j]);
    printf("PFit[%d] : ", j);
    printf("%2.4f \n", pSwarm -> particle[i].XFit);
  }

  return 0;
}

void ShowSwarm(SWARM *pSwarm)
{
  unsigned int i;
  //Para todas las partículas
  for(i = 0; i < pSwarm -> numParticles; i++)
  {
    ShowParticle(pSwarm, i);
  }

  return 0;
}

void DeleteSwarm(SWARM *pSwarm)
{
  unsigned int i;

  for (i = 0; i < pSwarm -> numParticles; i++)
  {
    free(pSwarm -> particle[i].Xi);
    free(pSwarm -> particle[i].Vi);
    free(pSwarm -> particle[i].Pi);
  }

  free(pSwarm);

  return 0;
}
```

## Pruebas y resultados

El programa, por ahora, hace todo de manera automática. Por lo cual no es necesario ingresar ningún parámetro o argumento. Las pruebas sólo consistieron en revisar la correcta creación del enjambre, y para ello se imprime con la función *showSwarm(...)*



## Conclusión

Se está desarrollando el método paso por paso, y a pesar de que ésta primera parte es bastante simple es el pilar fundamental del algoritmo, pues estamos definiendo la unidad base del POS, las partículas y reuniéndolas en el ejambre. Es importante entender y tener bien estructurada esta parte, de lo contrario los despecfectos u errores que pueda tener afectarán negativamente el resto del algoritmo y conforme aumente la complejidad del código la detección y arreglo de fallas se hará cada vez más complicado.

En esta parte del código, además de crear las partículas y agruparlas en el enjambre, estamos inicializando sus valores. Comienzan con posiciones aleatorias en el espacio y con esa posición se procede a evaluar la ecuación para obtener el peso de la partícula en ese instante de tiempo, en este caso la escuación está definida por código pero en un futuro se puede modificar para que el usuario ingrese la ecuación a analizar por el método POS. Los resultados sólo son muestras de los números asignados aleatoriamente a las partículas, pero conforme avance el algortimo estos número tomarán un mayor significado.