

División de Inenierías Campus Irapuato Salamanca
Universidad de Guanajuato Clase: Inteligencia artificial

Profesor: Dr. Carlos Hugo García Capulín
Estudiante: Brandon Marquez Salazar

Tarea 3

Uso de PSO Velocity Clamping

Entrega 12 de Mayo del 2023

Introducción

La optimización de enjambre de partículas (PSO) se considera importante en la inteligencia basada en enjambre. El PSO está relacionado con el estudio de los enjambres; donde se trata de una simulación de bandadas de pájaros. Se puede utilizar para resolver una amplia variedad de problemas de optimización. Es un método heurístico el cual, esencialmente, busca la solución a un modelo, basado en elementos específicos y un conjunto de estructuras, que emulan el comportamiento de un conjunto de individuos quienes buscan una posición específica, que se evalúa según el modelo planteado.

Planteamiento del problema

El problema que se nos plantea es lograr los resultados d'un experimento realizado para verificar l'eficiencia de PSO con valor de constricción, utilizando como función objetivo, alguno de los planteados'n la tabla 'TABLE I'. De la cual, se seleccionó la f2. Se tendrá que correr el programa 30 veces, la configuración utilizada es:

- Número de partículas: 30
- Valor C1: 30
- Valor C2: 30
- Número de iteraciones: 20'000
- Número de dimensiones: 100

TABLE I

NUMERICAL BENCHMARK FUNCTIONS WITH A VARYING NUMBER OF DIMENSIONS (DIM). REMARKS: 1) FUNCTIONS SINE AND COSINE OF THE ARGUMENTS IN RADIANS. 2) THE NOTATION $(a)^T(b)$ DENOTES THE DOT PRODUCT BETWEEN VECTORS a AND b . 3) THE FUNCTION u AND THE COEFFICIENTES REFERRED TO IN f_{12} AND f_{13} ARE GIVEN BY $u(x, a, b, c) = b(x - a)^c$ IF $x > a$, $u(x, a, b, c) = b(-x - a)^c$ IF $x < a$, $u(x, a, b, c) = 0$ IF $|x| \leq a$, AND FINALLY $y_i = 1 + \frac{1}{4}(x_i + 1)$. THE MATRIX a USED IN f_{14} , THE VECTORS a AND b USED IN f_{15} , AND THE MATRIX a AND THE VECTOR c USED IN $f_{21}-f_{23}$, ARE ALL DEFINED IN THE APPENDIX.

Function	Dim	Ranges	Minimum value
$f_1(\vec{x}) = \sum_{i=0}^{n-1} x_i^2$	30/100	$-5.12 \leq x_i \leq 5.12$	$f_1(\vec{0}) = 0$
$f_2(\vec{x}) = \sum_{i=0}^{n-1} x_i + \prod_{i=0}^{n-1} x_i$	30/100	$-10 \leq x_i \leq 10$	$f_2(\vec{0}) = 0$
$f_3(\vec{x}) = \sum_{i=0}^{n-1} \left(\sum_{j=0}^i x_j \right)^2$	30/100	$-100 \leq x_i \leq 100$	$f_3(\vec{0}) = 0$
$f_4(\vec{x}) = \max x_i , 0 \leq i < n$	30/100	$-100 \leq x_i \leq 100$	$f_4(\vec{0}) = 0$
$f_5(\vec{x}) = \sum_{i=0}^{n-1} (100 \cdot (x_{i+1} - (x_i)^2)^2 + (x_i - 1)^2)$	30/100	$-30 \leq x_i \leq 30$	$f_5(\vec{1}) = 0$
$f_6(\vec{x}) = \sum_{i=0}^{n-1} \left(\lfloor x_i + \frac{1}{2} \rfloor \right)^2$	30/100	$-100 \leq x_i \leq 100$	$f_6(\vec{p}) = 0,$ $-\frac{1}{2} \leq p_i < \frac{1}{2}$
$f_7(\vec{x}) = \left(\sum_{i=0}^{n-1} (i+1) \cdot x_i^4 \right) + \text{rand}[0, 1[$	30/100	$-1.28 \leq x_i \leq 1.28$	$f_7(\vec{0}) = 0$
$f_8(\vec{x}) = \sum_{i=0}^{n-1} -x_i \cdot \sin(\sqrt{ x_i })$	30/100	$-500 \leq x_i \leq 500$	$f_8(420.97) =$ $12569.5/41898.3$
$f_9(\vec{x}) = \sum_{i=0}^{n-1} (x_i^2 - 10 \cos(2\pi x_i) + 10)$	30/100	$-5.12 \leq x_i \leq 5.12$	$f_9(\vec{0}) = 0$
$f_{10}(\vec{x}) = -20 \exp \left(-0.2 \sqrt{\frac{1}{n} \sum_{i=0}^{n-1} x_i^2} \right) - \exp \left(\frac{1}{n} \sum_{i=0}^{n-1} \cos(2\pi x_i) \right) + 20 + e$	30/100	$-32 \leq x_i \leq 32$	$f_{10}(\vec{0}) = 0$
$f_{11}(\vec{x}) = \frac{1}{4000} \left(\sum_{i=0}^{n-1} x_i^2 \right) + \left(\prod_{i=0}^{n-1} \cos \left(\frac{x_i}{\sqrt{i+1}} \right) \right) + 1$	30/100	$-600 \leq x_i \leq 600$	$f_{11}(\vec{0}) = 0$
$f_{12}(\vec{x}) = \sum_{i=0}^{n-2} \left((y_i - 1)^2 (1 + 10 (\sin(\pi y_{i+1}))^2) + (y_n - 1)^2 \right) + \sum_{i=0}^{n-1} u(x_i, 10, 100, 4)$	30/100	$-50 \leq x_i \leq 50$	$f_{12}(-1) = 0$
$f_{13}(\vec{x}) = 0.1 \{ (\sin(3\pi x_1))^2 + \sum_{i=0}^{n-2} \left((x_i - 1)^2 (1 + (\sin(3\pi x_{i+1}))^2) \right) + (x_n - 1) (1 + (\sin(2\pi x_n))^2) \} + \sum_{i=0}^{n-1} u(x_i, 5, 100, 4)$	30/100	$-50 \leq x_i \leq 50$	$f_{13}(1, \dots, 1, -4.76) = -1.1428$
$f_{14}(\vec{x}) = \left(\frac{1}{500} + \sum_{j=0}^{24} (j+1 + \sum_{i=0}^1 (x_i - a_{ij})^6)^{-1} \right)^{-1}$	2	$-65.54 \leq x_i \leq 65.54$	$f_{14}(-31.95) = 0.998$
$f_{15}(\vec{x}) = \sum_{i=0}^{10} \left(a_i - \frac{x_0(b_i^2 + b_i x_1)}{b_i^2 + b_i x_2 + x_3} \right)^2$	4	$-5 \leq x_i \leq 5$	$f_{15}(0.19, 0.19, 0.12, 0.14) = 0.0003075$
$f_{16}(\vec{x}) = 4x_0^2 - 2.1x_0^4 + \frac{1}{3}x_0^6 + x_0x_1 - 4x_1^2 + 4x_1^4$	2	$-5 \leq x_i \leq 5$	$f_{16}(-0.09, 0.71) = -1.0316$
$f_{17}(\vec{x}) = (x_1 - \frac{5.1}{4\pi^2}x_0^2 + \frac{5}{\pi}x_0 - 6)^2 + 10(1 - \frac{1}{8\pi}) \cos(x_0) + 10$	2	$-5 \leq x_i \leq 15$	$f_{17}(9.42, 2.47) = 0.398$
$f_{18}(\vec{x}) = \{1 + (x_0 + x_1 + 1)^2 (19 - 14x_0 + 3x_0^2 - 14x_1 + 6x_0x_1 + 3x_1^2)\} \{30 + (2x_0 - 3x_1)^2 (18 - 32x_0 + 12x_0^2 + 48x_1 - 36x_0x_1 + 27x_1^2)\}$	2	$-2 \leq x_i \leq 2$	$f_{18}(1.49e-05, 1.00) = 3$
$f_{21}(\vec{x}) = -\sum_{i=0}^4 \left((x - a_i)^T (x - a_i) + c_i \right)^{-1}$	4	$0 \leq x_i \leq 10$	$f_{21}(\approx \vec{4}) = -10.2$
$f_{22}(\vec{x}) = -\sum_{i=0}^6 \left((x - a_i)^T (x - a_i) + c_i \right)^{-1}$	4	$0 \leq x_i \leq 10$	$f_{22}(\approx \vec{4}) = -10.4$
$f_{23}(\vec{x}) = -\sum_{i=0}^9 \left((x - a_i)^T (x - a_i) + c_i \right)^{-1}$	4	$0 \leq x_i \leq 10$	$f_{23}(\approx \vec{4}) = -10.5$

Descripción del programa

El programa consta de una estructura PARTICULA y una estructura ENJAMBRE. La estructura partícula tiene los valores importantes de posición y evaluación respecto al modelo matemático. También, se tienen funciones que permiten la operación de los elementos del enjambre, su inicialización, actualización e interacción. El código está dividido en una cabecera que define las operaciones esenciales del algoritmo, las estructuras y el prototipo de las dos funciones q'el usuario puede definir: función objetivo (el cual albergará al modelo matemático), función proceso (que albergará'l proceso que llevará, desde la creación del enjambre, las evaluaciones, entre otros elementos del procesamiento, hasta su limpieza).

Código utilizado

Cabecera

```
#ifndef __pso_header__
#define __pso_header__

// Definición de la estructura Partícula
// Esta partícula representa a un individuo
// El individuo buscará tener la mejor posición
// Habrá dos criterios: En el primero, ve la mejor
// posición que ha tenido durante su existencia; en
// el segundo, ve qué partícula tiene la mejor posición
// en ese momento.
// Recalcula su valor de paso (velocidad) y, a partir de
// ahí, suma el paso a su posición actual, obteniendo el
// nuevo valor de posición.
// La partícula requiere saber en cuántas dimensiones
// estará moviéndose. Dichas dimensiones definirán al vector
// posición y al vector velocidad.
typedef struct {
    float *Xi;      //Posicion
    float *Vi;      //Velocidad
    float *Pi;      //Mejor Posicion Historica
    float Xfit;    //Valor de Fitnes para la posicion actual
    float Pfit;    //Valor de Fitnes para la Mejor Posicion Historica
```

```

}PARTICULA;

// Definición de la estructura Enjambre
// El enjambre es un conjunto de partículas
// Este conjunto actuará para encontrar soluciones
// Cada solución es repensada según los valores históricos
// y valores presentes.

typedef struct{
    PARTICULA *Part;                                // Partículas
    unsigned int CantidadDeParticulas;             // Número de partículas
    unsigned int CantidadDeParametros;            // Número de parámetros del
                                                problema
    unsigned int MejorParticulaDelGrupo;          // ID de la mejor partícula del
                                                grupo
    unsigned int MaximoDeIteraciones;            // Número máximo d'iteraciones
                                                a realizar
    float C1;                                     // Valor de peso C1
    float C2;                                     // Valor de peso C2
    const float *LimitesSuperiores;              // Limites Superiores
    const float *LimitesInferiores;              // Limites Inferiores
    float X;                                      // Factor de restricción (
                                                convergencia)
    float Constriccion;                          // Factor de restricción (
                                                convergencia)
}ENJAMBRE;
}

// Operadores del enjambre (métodos)

/* Creador de enjambres:
* Recibe el número de partículas y el número de parámetros
* (variables del problema)*/
ENJAMBRE* CrearEnjambre(
    //ENJAMBRE      *--Enjambre-- ,
    unsigned int --CantidadDeParticulas-- ,
    unsigned int --CantidadDeParametros-- )
/* Inicializador de enjambres:
* Define los valores predeterminados (de inicio), de los individuos.
* Recibe el enjambre, la posición inicial, las variables del problema,
* y los límites*/
void InicializarEnjambre(
    ENJAMBRE      *--Enjambre-- ,
    float         --FactorConstriccion-- ,
    float         --ValorDePeso_C1-- ,
    float         --ValorDePeso_C2-- ,
    unsigned int --MaximoDeIteraciones-- ,
    const float *--LimitesInferiores-- ,
    const float *--LimitesSuperiores-- )
/* Una vez terminado el programa, ésta función liberará la memoria

```

```

* que se reservó durante la creación del enjambre. Como argumento,
* recibe al apuntador del enjambre.*/
void EliminarEnjambre(
    ENJAMBRE *__Enjambre__
);
/* Nos permite visualizar los parámetros de la partícula.*/
void ImprimeParticulaID(
    ENJAMBRE      *__Enjambre__ ,
    unsigned int   __ID_Particula__
);
/*Imprime la particula sin enjambre*/
void ImprimeParticula(
    const PARTICULA      __Particula__ ,
    const unsigned int   __CantidadDeParametros__
);
/* Permite visualizar los parámetros del enjambre, y las partículas
 * que le componen.*/
void ImprimeEnjambre(
    ENJAMBRE *__Enjambre__
);
/* Permite valorar al enjambre, según los criterios del PSO
 * y de la función objetivo*/
void EvaluarEnjambreMin(
    ENJAMBRE      __Enjambre__ ,
    const float   __ParametrosDeOperacion__
);
void EvaluarEnjambreMax(
    ENJAMBRE      __Enjambre__ ,
    const float   __ParametrosDeOperacion__
);
/* Similar a EvaluarEnjambre, con la particularidad de que Inicializa
 * los valores de Mejor Posicion Historica , de las partículas*/
void EvaluacionInicialEnjambreMin(
    ENJAMBRE      __Enjambre__ ,
    const float   __ParametrosDeOperacion__
);
void EvaluacionInicialEnjambreMax(
    ENJAMBRE      __Enjambre__ ,
    const float   __ParametrosDeOperacion__
);
/* Renueva la velocidad basado en los vectores de
 * Posición Actual ,
 * Mejor Posicion Historica y
 * Mejor Posicion Global Actual*/
void ActualizarVelocidad(
    ENJAMBRE *__Enjambre__
);
void ActualizarVelocidadInerciaW(
    ENJAMBRE *__Enjambre__
);
void ActualizarVelocidadConstriction(

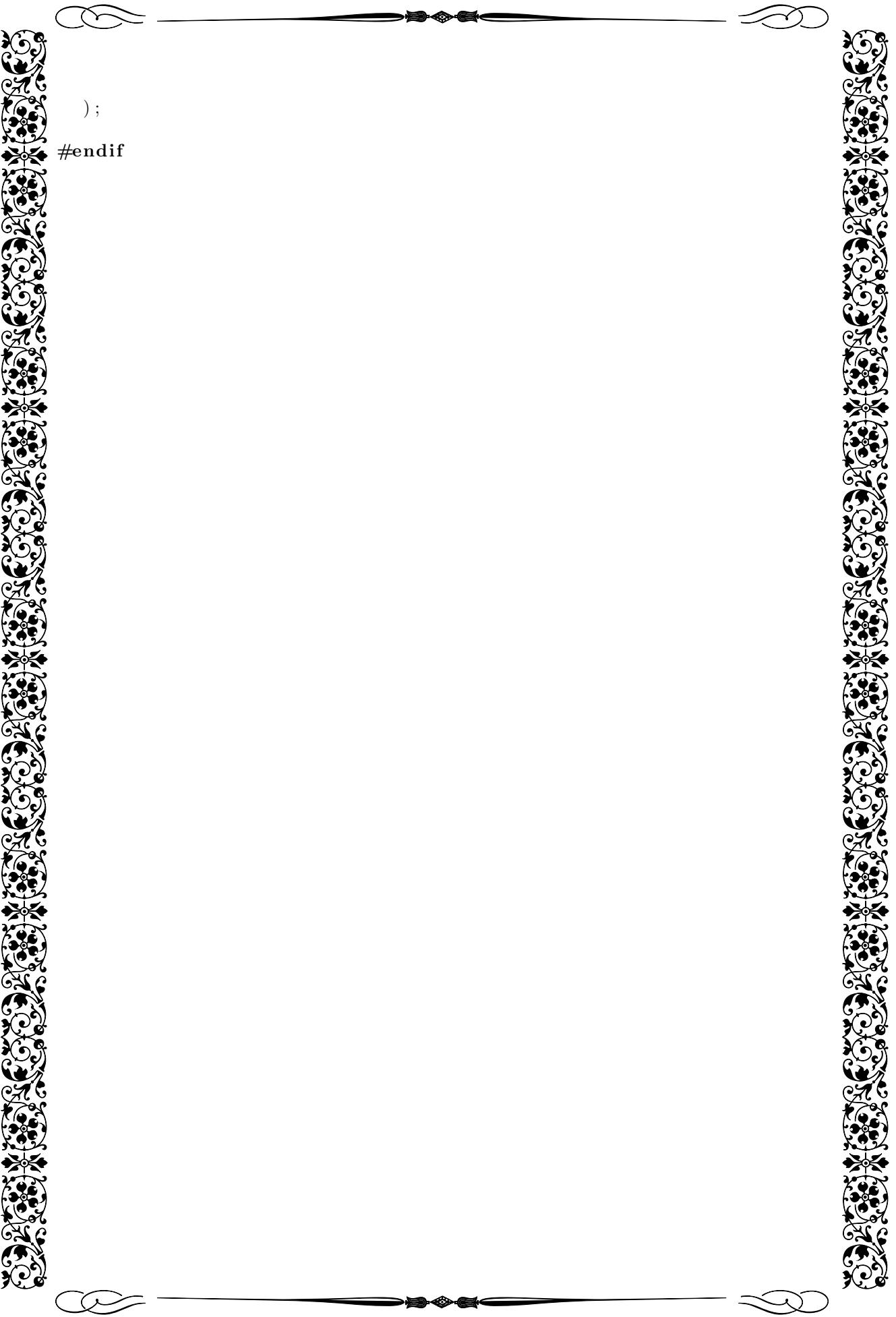
```

```

    ENJAMBRE *--Enjambre--
);
/* Suma los valores de velocidad a la posición actual de
 * cada partícula.*/
void ActualizarPosicion(
    ENJAMBRE *--Enjambre--
);
/* Valora, en cada partícula, si el valor actual es mejor que'l mejor
 * valor histórico; si los valores actuales son mejores, actualiza
 * los parámetros.*/
void ActualizarMejoresPosicionesMin(
    ENJAMBRE *--Enjambre--
);
void ActualizarMejoresPosicionesMax(
    ENJAMBRE *--Enjambre--
);

/* La función a evaluar, regresa el valor de fitness (precisión)
 * Requiere ser definida para l'evaluación d'as partículas
 * Valores De Parámetros .... (arreglo float)
 * Cantidad De Parámetros ... (int)
 * Parámetros De Operación .. (arreglo float)
*/
float FuncionObjetivo(
    float          *--ValoresDeParametros-- ,
    unsigned int   --CantidadDeParametros-- ,
    const float   *--ParametrosDeOperacion-- )
;

/* Función que se puede definir para realizar el procesamiento pso,
 * puede ser ignorado o definido y consta d'os sig. elementos, en
 * el orden en que se presentan a continuación:
 * Número De Partículas ..... (float)
 * Dimensión ..... (float)
 * Límites Superiores ..... (arreglo float)
 * Límite Inferior ..... (arreglo float)
 * Número Máximo De Iteraciones ..... (int)
 * Factor De Constricción O De Inercia .. (float)
 * Valor Peso C1: Mejor Personal ..... (float)
 * Valor Peso C2: Mejor Global ..... (float)
 * Parámetros De Operación ..... (arreglo float) */
PARTICULA ProcesoPSO(
    const float      --NúmeroDePartículas-- ,
    const float      --Dimensión-- ,
    const float      *--LímiteSuperior-- ,
    const float      *--LímiteInferior-- ,
    const unsigned int --NúmeroMáximoDeIteraciones-- ,
    const float      --Factor_Constricción_Inercia-- ,
    const float      --ValorPesoPersonalC1-- ,
    const float      --ValorPesoGlobalC2-- ,
    const float      *--ParámetrosDeOperación-- 
```



Definiciones

```
#include "pso.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

// Definición de las funciones

ENJAMBRE* CrearEnjambre(
    //ENJAMBRE* __Enjambre__ ,
    unsigned int __CantidadDeParticulas__ ,
    unsigned int __CantidadDeParametros__
){
    ENJAMBRE *ptr=NULL;
    //Reservar la memoria para la estructura del enjambre
    ptr=(ENJAMBRE *) malloc(sizeof(ENJAMBRE));
    if(ptr==NULL){
        printf("Error al reservar la memoria para la estructura ENJAMBRE.");
        ;
        exit(0);
    }
    ptr->CantidadDeParticulas=__CantidadDeParticulas__;
    ptr->CantidadDeParametros=__CantidadDeParametros__;

    //Reservar la memoria para N particulas de M parametros
    ptr->Part=NULL;
    ptr->Part=(PARTICULA *) malloc(__CantidadDeParticulas__*sizeof(
        PARTICULA));
    if(ptr->Part==NULL){
        printf("Error al reservar la memoria para las Particulas.");
        exit(0);
    }
    //Reservar memoria para los 3 vectores de cada Particula
    for(unsigned int i=0; i<__CantidadDeParticulas__; ++i){
        ptr->Part[i].Xi=(float *) malloc(__CantidadDeParametros__*sizeof(
            float));
        ptr->Part[i].Vi=(float *) malloc(__CantidadDeParametros__*sizeof(
            float));
        ptr->Part[i].Pi=(float *) malloc(__CantidadDeParametros__*sizeof(
            float));
    }
    return ptr;
}

void InicializarEnjambre(
    ENJAMBRE     *__Enjambre__ ,
    float        __FactorConstriccion__ ,
```

```

float      __ValorDePeso_C1__ ,
float      __ValorDePeso_C2__ ,
unsigned int __MaximoDeIteraciones__ ,
const float *__LimitesInferiores__ ,
const float *__LimitesSuperiores__ 

){

if( __Enjambre__ ){

float aux , rango ;
__Enjambre__->X          = __FactorConstriccion__ ;
__Enjambre__->C1          = __ValorDePeso_C1__ ;
__Enjambre__->C2          = __ValorDePeso_C2__ ;
__Enjambre__->MaximoDeIteraciones = __MaximoDeIteraciones__ ;
__Enjambre__->MejorParticulaDelGrupo = 0;
__Enjambre__->LimitesInferiores    = __LimitesInferiores__ ;
__Enjambre__->LimitesSuperiores    = __LimitesSuperiores__ ;
//Dar constriccion uwu
float fi = __Enjambre__->C1+__Enjambre__->C2;
__Enjambre__->Constriccion=2/fabs(2-fi-sqrt(pow( fi ,2)-(4* fi )) );
//printf("%f\n", fi );
//printf("%f",pow( fi ,2)-(4* fi ) );
//printf("%f",sqrt(pow( fi ,2)-(4* fi )) );
//printf("%f",fabs(2-fi-sqrt(pow( fi ,2)-(4* fi ))));
//printf("%f\n\n",__Enjambre__->Constriccion);
//Inicializar cada vector de cada particula
for(unsigned int i=0; i<__Enjambre__->CantidadDeParticulas; ++i) //
    Para cada particula i
    for(unsigned int j=0; j<__Enjambre__->CantidadDeParametros; ++j) //
        Para cada parametro j de cada vector de la particula i
    { rango=__Enjambre__->LimitesSuperiores[ j ]-__Enjambre__->
        LimitesInferiores[ j ];
    aux= ((float)rand()/(float)RAND_MAX) * rango + __Enjambre__->
        LimitesInferiores[ j ];
    __Enjambre__->Part[ i ].Xi[ j ]=aux;
    __Enjambre__->Part[ i ].Vi[ j ]=0;
    __Enjambre__->Part[ i ].Pi[ j ]=aux;
    }
}
}

void EliminarEnjambre(ENJAMBRE* __Enjambre__ )
{ //Liberar la memoria para de los 3 vectores de cada Particula
    for(unsigned int i=0; i<__Enjambre__->CantidadDeParticulas; ++i)
    { free( __Enjambre__->Part[ i ].Xi );
        free( __Enjambre__->Part[ i ].Vi );
        free( __Enjambre__->Part[ i ].Pi );
    }
    //Liberar la memoria de las estructuras particula
    free( __Enjambre__->Part );
    //Liberar la memoria de la estructura del enajmbre
    free( __Enjambre__ );
}

```

```

}

void ImprimeParticulaID (ENJAMBRE *_Enjambre_ , unsigned int
    _ID_Particula_ ){
    printf ("\nP% ,Xi:_" ,_ID_Particula_ );
    for (unsigned int i=0; i<_Enjambre_>CantidadDeParametros; i++)
        printf (" %,_" ,_Enjambre_>Part [_ID_Particula_ ]. Xi[ i ] );
    printf ("\nP% ,Vi:_" ,_ID_Particula_ );
    for (unsigned int i=0; i<_Enjambre_>CantidadDeParametros; i++)
        printf (" %,_" ,_Enjambre_>Part [_ID_Particula_ ]. Vi[ i ] );
    printf ("\nP% ,Pi:_" ,_ID_Particula_ );
    for (unsigned int i=0; i<_Enjambre_>CantidadDeParametros; i++)
        printf (" %,_" ,_Enjambre_>Part [_ID_Particula_ ]. Pi[ i ] );
    printf ("\nP% ,Xfit=%f" ,_ID_Particula_ ,_Enjambre_>Part [
        _ID_Particula_ ]. Xfit );
    printf ("\nP% ,Pfit=%f" ,_ID_Particula_ ,_Enjambre_>Part [
        _ID_Particula_ ]. Pfit );
}

void ImprimeParticula(
    const PARTICULA    *_Particula_ ,
    const unsigned int   _CantidadDeParametros_
){
    printf ("\nParticula:");
    for (unsigned int i=0; i<_CantidadDeParametros; i++)
        printf (" %,_" ,_Particula->Xi[ i ] );
    printf ("\nParticula_Vi:_");
    for (unsigned int i=0; i<_CantidadDeParametros; i++)
        printf (" %,_" ,_Particula->Vi[ i ] );
    printf ("\nParticula_Pi:_");
    for (unsigned int i=0; i<_CantidadDeParametros; i++)
        printf (" %,_" ,_Particula->Pi[ i ] );
    printf ("\nParticula_Xfit=%f" ,_Particula->Xfit );
    printf ("\nParticula_Pfit=%f" ,_Particula->Pfit );
}

void ImprimeEnjambre (ENJAMBRE *_Enjambre_ )
{ for (unsigned int i=0; i<_Enjambre->CantidadDeParticulas; ++i) //  

    Para cada particula i  

    ImprimeParticulaID (_Enjambre_ , i );
}

void EvaluarEnjambreMin (ENJAMBRE *_Enjambre_ , const float*
    _ParametrosDeOperacion_ ){
    float BestFit;
    // Calcular el valor de Fitness de cada particula
    BestFit = FuncionObjetivo (
        _Enjambre->Part [0]. Xi ,
        _Enjambre->CantidadDeParametros ,

```

```

    --ParametrosDeOperacion_
);
for(unsigned int i=0; i<_Enjambre->CantidadDeParticulas; i++){
    _Enjambre->Part[i].Xfit = FuncionObjetivo(
        _Enjambre->Part[i].Xi,
        _Enjambre->CantidadDeParametros ,
        _ParametrosDeOperacion_
    );
    // Almacena el indice de la mejor particula de todo en enjambre
    if(_Enjambre->Part[i].Xfit<BestFit){
        BestFit = _Enjambre->Part[i].Xfit;
        _Enjambre->MejorParticulaDelGrupo =i;
    }
}

void EvaluarEnjambreMax(ENJAMBRE *_Enjambre_,const float*
    _ParametrosDeOperacion_){
    float BestFit;
    // Calcular el valor de Fitness de cada particula
    BestFit = FuncionObjetivo(
        _Enjambre->Part[0].Xi,
        _Enjambre->CantidadDeParametros ,
        _ParametrosDeOperacion_
    );
    for(unsigned int i=0; i<_Enjambre->CantidadDeParticulas; i++){
        _Enjambre->Part[i].Xfit = FuncionObjetivo(
            _Enjambre->Part[i].Xi,
            _Enjambre->CantidadDeParametros ,
            _ParametrosDeOperacion_
        );
        // Almacena el indice de la mejor particula de todo en enjambre
        if(_Enjambre->Part[i].Xfit>BestFit){
            BestFit = _Enjambre->Part[i].Xfit;
            _Enjambre->MejorParticulaDelGrupo =i;
        }
    }
}

void EvaluacionInicialEnjambreMin(ENJAMBRE *_Enjambre_,const float*
    _ParametrosDeOperacion_){
    if(_Enjambre_){
        float aux , BestFit;
        //Calcular el valor de fitness de cada Particula
        BestFit=FuncionObjetivo(
            _Enjambre->Part[0].Xi,
            _Enjambre->CantidadDeParametros ,
            _ParametrosDeOperacion_
        );
        for(unsigned int i=0; i<_Enjambre->CantidadDeParticulas; i++){

```

```

aux=FuncionObjetivo(
    __Enjambre__->Part [ i ]. Xi ,
    __Enjambre__->CantidadDeParametros ,
    __ParametrosDeOperacion__
);
__Enjambre__->Part [ i ]. Xfit=aux;
__Enjambre__->Part [ i ]. Pfit=aux;
//Almacena el indice de la mejor particula de todo el enjambre
if(aux<BestFit){
    BestFit=aux;
    __Enjambre__->MejorParticulaDelGrupo=i;
}
}

void EvaluacionInicialEnjambreMax (ENJAMBRE * __Enjambre__ ,const float *
__ParametrosDeOperacion__){
if( __Enjambre__){
float aux , BestFit;
//Calcular el valor de fitness de cada Particula
BestFit=FuncionObjetivo(
    __Enjambre__->Part [ 0 ]. Xi ,
    __Enjambre__->CantidadDeParametros ,
    __ParametrosDeOperacion__
);
for(unsigned int i=0; i<__Enjambre__->CantidadDeParticulas ; i++){
    aux=FuncionObjetivo(
        __Enjambre__->Part [ i ]. Xi ,
        __Enjambre__->CantidadDeParametros ,
        __ParametrosDeOperacion__
    );
    __Enjambre__->Part [ i ]. Xfit=aux;
    __Enjambre__->Part [ i ]. Pfit=aux;
    //Almacena el indice de la mejor particula de todo el enjambre
    if(aux>BestFit){
        BestFit=aux;
        __Enjambre__->MejorParticulaDelGrupo=i;
    }
}
}

void ActualizarVelocidad (ENJAMBRE * __Enjambre__){
float Y1,Y2;
//Actualizar cada vector velocidad Vi de cada particula
for(unsigned int i=0; i<__Enjambre__->CantidadDeParticulas ; i++) // Para cada particula i
    for(unsigned int j=0; j<__Enjambre__->CantidadDeParametros ; j++) // Para cada parametro j de cada vector Vi de la particula i

```

```

{
    Y1=rand()/(float)RANDMAX;
    Y2=rand()/(float)RANDMAX;
    __Enjambre__->Part [ i ]. Vi [ j ] =(
        __Enjambre__->Part [ i ]. Vi [ j ]+
        ( __Enjambre__->C1*Y1*( __Enjambre__->Part [ i ]. Pi [ j ]-__Enjambre__->Part [ i ]. Xi [ j ]) +
        ( __Enjambre__->C2*Y2*( __Enjambre__->Part [ __Enjambre__->MejorParticulaDelGrupo ]. Pi [ j ]-__Enjambre__->Part [ i ]. Xi [ j ]) )
    );
}
}

void ActualizarVelocidadInerciaW (ENJAMBRE *__Enjambre__){
    float Y1,Y2;
    //Actualizar cada vector velocidad Vi de cada particula
    for(unsigned int i=0; i<__Enjambre__->CantidadDeParticulas; i++) // Para cada particula i
        for(unsigned int j=0; j<__Enjambre__->CantidadDeParametros; j++) // Para cada parametro j de cada vector Vi de la particula i
    {
        Y1=rand()/(float)RANDMAX;
        Y2=rand()/(float)RANDMAX;
        __Enjambre__->Part [ i ]. Vi [ j ] =(
            ( __Enjambre__->Part [ i ]. Vi [ j ]* __Enjambre__->X)+
            ( __Enjambre__->C1*Y1*( __Enjambre__->Part [ i ]. Pi [ j ]-__Enjambre__->Part [ i ]. Xi [ j ]) +
            ( __Enjambre__->C2*Y2*( __Enjambre__->Part [ __Enjambre__->MejorParticulaDelGrupo ]. Pi [ j ]-__Enjambre__->Part [ i ]. Xi [ j ]) )
        );
    }
}

void ActualizarVelocidadConstriction (
    ENJAMBRE *__Enjambre__
)
{
    float Y1,Y2;
    //Actualizar cada vector velocidad Vi de cada particula
    for(unsigned int i=0; i<__Enjambre__->CantidadDeParticulas; i++) // Para cada particula i
        for(unsigned int j=0; j<__Enjambre__->CantidadDeParametros; j++) // Para cada parametro j de cada vector Vi de la particula i
    {
        Y1=rand()/(float)RANDMAX;
        Y2=rand()/(float)RANDMAX;
        //printf("Y1= %f Y2= %f", Y1, Y2);
        __Enjambre__->Part [ i ]. Vi [ j ] =__Enjambre__->Constriccion*(
            ( __Enjambre__->Part [ i ]. Vi [ j ]+
            ( __Enjambre__->C1*Y1*( __Enjambre__->Part [ i ]. Pi [ j ]-__Enjambre__->Part [ i ]. Xi [ j ]) +
            ( __Enjambre__->C2*Y2*( __Enjambre__->Part [ __Enjambre__->MejorParticulaDelGrupo ]. Pi [ j ]-__Enjambre__->Part [ i ]. Xi [ j ]) )
        );
    }
}

```

```

        ));  

        //printf(" Vt+1= %f ",--Enjambre__->Part[i].Vi[j]);  

    }  

}  
  

void ActualizarPosicion (ENJAMBRE *--Enjambre__){  

    // Acutailzsr cada vector Posicion XI de cada particula  

    for (unsigned int i=0; i<--Enjambre__->CantidadDeParticulas; i++) //  

        Para cada particula i  

    for (unsigned int j=0; j<--Enjambre__->CantidadDeParametros; j++) //  

        Para cada parametro j de cada vector de la particula i  

        --Enjambre__->Part[i].Xi[j] += --Enjambre__->Part[i].Vi[j];  

}  
  

void ActualizarMejoresPosicionesMin (ENJAMBRE *--Enjambre__){  

    for (unsigned int i=0; i<--Enjambre__->CantidadDeParticulas; i++)  

        if (--Enjambre__->Part[i].Xfit < --Enjambre__->Part[i].Pfit){  

            --Enjambre__->Part[i].Pfit = --Enjambre__->Part[i].Xfit;  

        for (unsigned int j=0; j<--Enjambre__->CantidadDeParametros; j++)  

            //Para cada parametro j de cada vector de la particula i  

            --Enjambre__->Part[i].Pi[j] = --Enjambre__->Part[i].Xi[j];  

    }  

}  
  

void ActualizarMejoresPosicionesMax (ENJAMBRE *--Enjambre__){  

    for (unsigned int i=0; i<--Enjambre__->CantidadDeParticulas; i++)  

        if (--Enjambre__->Part[i].Xfit > --Enjambre__->Part[i].Pfit){  

            --Enjambre__->Part[i].Pfit = --Enjambre__->Part[i].Xfit;  

        for (unsigned int j=0; j<--Enjambre__->CantidadDeParametros; j++)  

            //Para cada parametro j de cada vector de la particula i  

            --Enjambre__->Part[i].Pi[j] = --Enjambre__->Part[i].Xi[j];  

    }  

}  
  

/*float FuncionObjetivo (float *_ValoresDeParametros__, unsigned int  

    _CantidadDeParametros__){  

    unsigned int k;  

    float fit , aux = 0;  

    // Maximar la siguiente funcion:  

    // f(x,y)=50-(x-5)^2-(y-5)^2;  

    // fit=250-pow(Xi[0]+7,2)-pow(Xi[1]-3,2)-pow(Xi[2]-3,2)-pow(Xi[3]-5,2)  

    // -pow(Xi[4]-8,2);  

    // Funcion Rastriging (Buscamos el valor 0)  

    for (k=0; k<--CantidadDeParametros__; k++)  

        aux += pow(--ValoresDeParametros__[k],2)-10*cos(6.283185*  

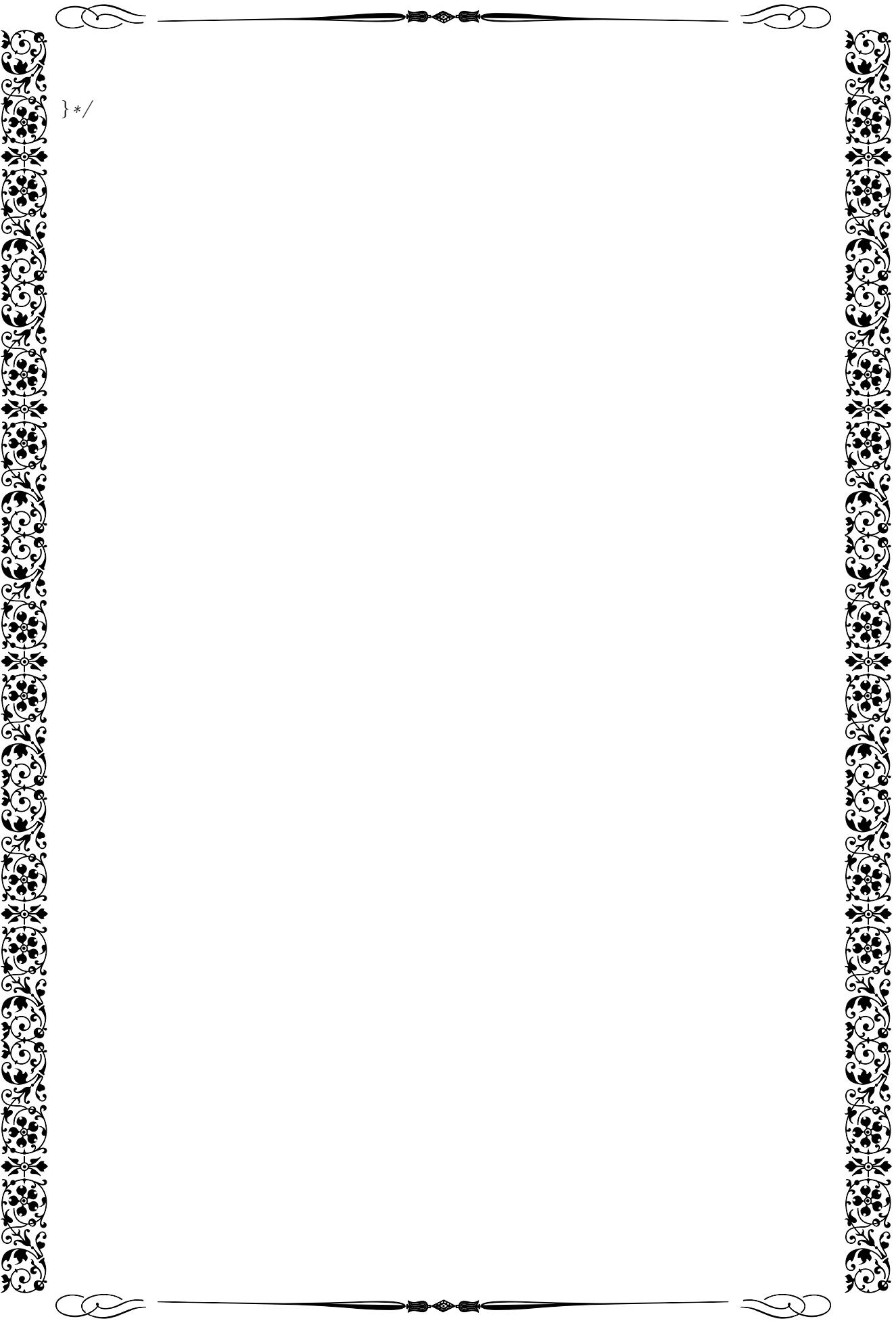
            --ValoresDeParametros__[k])+10;  

    fit = 100 - aux; // El valor d'a precision se ponderara en una escala  

    del 1 al 100  

    return fit;
}

```



Programa principal, proceso pso

```
#include <stdio.h>
#include <math.h>
#include "../ pso.h"
#include <time.h>
#include <stdlib.h>

// The update function uses the maximum value , so , if we search for the
// minimum value ,
// then we need to invert the result of FuncionObjetivo .

void FuncionObjetivoF(
    float      *__ValoresDeParametros__ ,
    unsigned int __CantidadDeParametros__ ,
    const float *__ResultadoDeOperacion__
);

int main (void){
    //Programa que obtenga los valores
    // Limites={InfX, InfY, SupX, SupY}
    PARTICULA LaSelecta ;
    unsigned int Dimension=100,k=0;
    float *LimitesInf=(float*) malloc (Dimension*sizeof(float));
    float *LimitesSup=(float*) malloc (Dimension*sizeof(float));
    while (k<Dimension) {
        *(LimitesInf+k)=-10;
        *(LimitesSup+k)=10;
        ++k;
    }

    LaSelecta= ProcesoPSO (
        100 , Dimension ,
        LimitesSup , LimitesInf ,
        20000 , 0.6 , 2.08 , 2.08 ,
        NULL
    );
    //ImprimeParticula(&LaSelecta , Dimension);
    printf("%i,%f\n" ,LaSelecta . Xfit , LaSelecta . Pfit );
    free(LaSelecta . Pi );
    free(LaSelecta . Vi );
    free(LaSelecta . Xi );
    free(LimitesInf );
    free(LimitesSup );
}

/*
-----*/
```

```

float FuncionObjetivo(
    float      *--ValoresDeParametros-- ,
    unsigned int --CantidadDeParametros-- ,
    const float *--ParametrosDeOperacion-- )
{
    float ResultadoDeOperacion=0;
    FuncionObjetivoF(
        --ValoresDeParametros-- , --CantidadDeParametros-- ,
        &ResultadoDeOperacion
    );
    return fabs(ResultadoDeOperacion);
}

PARTICULA ProcesoPSO(
    const float      --NumeroDeParticulas-- ,
    const float      --Dimension-- ,
    const float      *--LimiteSuperior-- ,
    const float      *--LimiteInferior-- ,
    const unsigned int --NumeroMaximoDeIteraciones-- ,
    const float      --Factor_Constriccion_Inercia-- ,
    const float      --ValorPesoPersonalC1-- ,
    const float      --ValorPesoGlobalC2-- ,
    const float      *--ParametrosDeOperacion-- )
{
    ENJAMBRE *Enj=NULL;
    PARTICULA Particle;
    srand(time(NULL));
    unsigned int t=0;

    //Crear un enjambre de NumeroParticulas de Numero de parametros igual
    //a Dimension
    Enj=CrearEnjambre(
        --NumeroDeParticulas-- ,
        --Dimension-- )
    ;

    InicializarEnjambre(
        Enj ,
        --Factor_Constriccion_Inercia-- ,
        --ValorPesoPersonalC1-- ,
        --ValorPesoGlobalC2-- ,
        --NumeroMaximoDeIteraciones-- ,
        --LimiteInferior-- ,
        --LimiteSuperior-- )
    ;

    EvaluacionInicialEnjambreMin(Enj , --ParametrosDeOperacion-- );

    while((t++)<Enj->MaximoDeIteraciones){
        ActualizarVelocidadConstriction(Enj);
}

```

```
ActualizarPosicion(Enj);  
EvaluarEnjambreMin(Enj, __ParametrosDeOperacion__);  
ActualizarMejoresPosicionesMin(Enj);  
}  
  
Particle . Xi=(Enj->Part+Enj->MejorParticulaDelGrupo)->Xi;  
(Enj->Part+Enj->MejorParticulaDelGrupo)->Xi=NULL;  
Particle . Vi=(Enj->Part+Enj->MejorParticulaDelGrupo)->Vi;  
(Enj->Part+Enj->MejorParticulaDelGrupo)->Vi=NULL;  
Particle . Pi=(Enj->Part+Enj->MejorParticulaDelGrupo)->Pi;  
(Enj->Part+Enj->MejorParticulaDelGrupo)->Pi=NULL;  
Particle . Xfit=(Enj->Part+Enj->MejorParticulaDelGrupo)->Xfit;  
Particle . Pfit=(Enj->Part+Enj->MejorParticulaDelGrupo)->Pfit;  
  
EliminarEnjambre(Enj);  
return Particle;  
}
```

Programa principal, definición de modelo matemático (fortran)

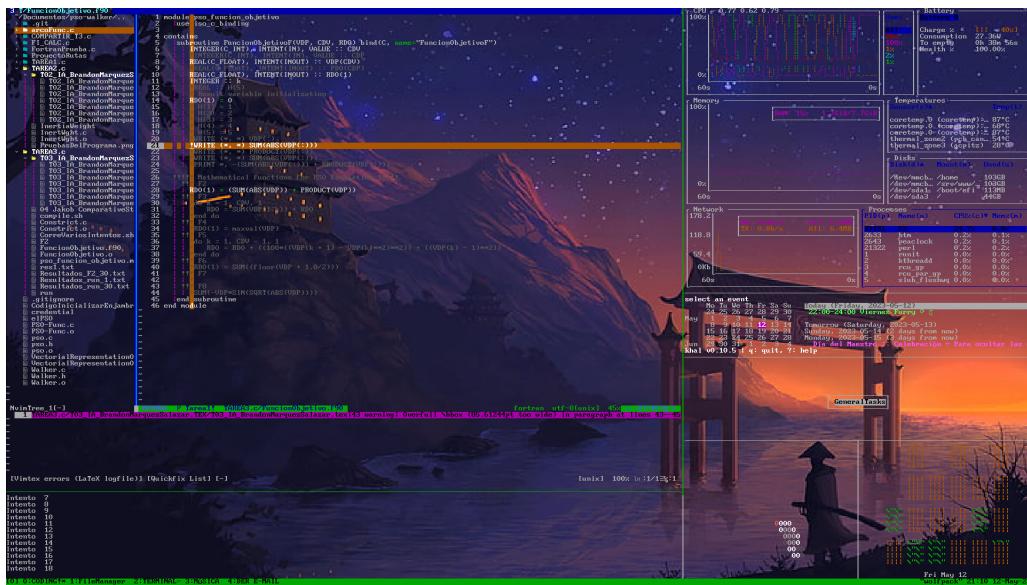
```
module pso_funcion_objetivo
  use iso_c_binding

contains
  subroutine FuncionObjetivoF(VDP, CDV, RDO) bind(C, name="FuncionObjetivoF")
    INTEGER(C_INT), INTENT(IN), VALUE :: CDV
    !INTEGER(C_INT), INTENT(IN), VALUE :: CDP
    REAL(C_FLOAT), INTENT(INOUT) :: VDP(CDV)
    !REAL(C_FLOAT), INTENT(INOUT) :: PDO(CDP)
    REAL(C_FLOAT), INTENT(INOUT) :: RDO(1)
    INTEGER :: k
    !REAL :: H(5)
    ! Result variable initialization
    RDO(1) = 0
    ! H(1) = 1
    ! H(2) = 2
    ! H(3) = 3
    ! H(4) = 4
    ! H(5) = 5
    !WRITE (*, *) VDP(:)
    !WRITE (*, *) SUM(ABS(VDP(:)))
    !WRITE (*, *) PRODUCT(VDP(:))
    !WRITE (*, *) SUM(ABS(VDP(:)))
    !PRINT *, -(SUM(ABS(VDP(:))) + PRODUCT(VDP(:)))

    !!!!! Mathematical Functions for PSO Evaluation !!!!!
    !!! F2
    !RDO(1) = (SUM(ABS(VDP)) + PRODUCT(VDP))
    !!! F3
    !do k = 1, CDV, 1
    !  RDO = SUM(VDP(1:k)) + RDO
    !end do
    !!! F4
    RDO(1) = maxval(VDP)
    !!! F5
    !do k = 1, CDV - 1, 1
    !  RDO = RDO + ((100*((VDP(k + 1) - VDP(k)**2)**2)) + ((VDP(k) - 1)**2))
    !end do
    !!! F6
    !RDO(1) = SUM((floor(VDP + 1.0/2)))
    !!! F7
    !
    !!! F8
```

```
!RDO(1) = SUM(-VDP*SIN(SQRT(ABS(VDP))))  
end subroutine  
end module
```

Pruebas y resultados



Datos de resultados

Local Best	Global Best
12.000000i	0.000000
15.000000i	0.000000
9.000000i	0.000000
17.000000i	0.000000
0.000000i	0.000000
0.000000i	0.000000
11.000000i	0.000000
4.000000i	0.000000
1.000000i	0.000000
39.000000i	0.000000
997.000000i	0.000000
31.000000i	0.000000
157.000000i	0.000000
28.000000i	0.000000
2.000000i	0.000000
1.000000i	0.000000
4.000000i	0.000000
28.000000i	0.000000
16.000000i	0.000000
2.000000i	0.000000
10.000000i	0.000000
24.000000i	0.000000
10.000000i	0.000000
2.000000i	0.000000
9.000000i	0.000000
12.000000i	0.000000
2.000000i	0.000000
6.000000i	0.000000
29.000000i	0.000000
4.000000i	0.000000

Con un promedio de: 49.4i para Local Best
y un promedio de 0 para Global Best
Sin poder calcular la DevEst

Conclusión

El PSO con restricción impide la dispersión de un grupo de individuos, sin embargo, sus características matemáticas nos impide, también, una selección libre de los elementos C1 y C2, ya que la ecuación puede dar indeterminación, si acaso los valores no corresponden adecuadamente a la ecuación de restricción. Algo muy importante de plantear, es que se tuvo que sintonizar adecuadamente para evitar resultados extremos. El correrlo más de una vez, permitió observar el comportamiento y, estimando el promedio, verificar la precisión del método.

