

UNIVERSIDAD DE GUANAJUATO

DIVISIÓN DE INGENIERÍAS
CAMPUS – IRAPUATO SALAMANCA

Inteligencia Artificial

PROF. GARCÍA CAPULÍN CARLOS HUGO

TAREA 6

Sintonización del algoritmo PSO

Martínez Lona Verónica Montserrat

Martes 7 Marzo, 2017

Introducción:

Hicimos la sintonización del algoritmo PSO y sus efectos en el desempeño. También analizamos las ventajas o desventajas de aplicar alguno de las opciones de sintonización.

La sintonización del algoritmo se constituye de:

- Número de parámetros de las partículas.
- Número de partículas en el enjambre.
- Límites para el espacio de búsqueda de las partículas.
- Condición de término (Número de iteraciones / Umbral de error).

Desarrollo:

Ya teniendo el algoritmo desarrollado nos dedicamos a hacer pruebas simples de rendimiento para ver de qué manera afectaba cambiar ciertos parámetros en el algoritmo, tales como el número de partículas o el margen de error.

Los números que escogimos y los límites fueron meramente sin pensar y en ocasiones algo demasiado grande para observar claramente el efecto que tenía sobre la eficiencia del método.

El desarrollo de la práctica consistió básicamente en modificar los siguientes valores:

```
const unsigned int NUMBER_PARTICLES;  
const unsigned int NUMBER_ITERATIONS;  
const float LimiteInf;  
const float LimiteSup;  
const float Vmin;  
const float Vmax;  
(50 - expSwarm -> particle[expSwarm -> idGbest].PFit) > error;
```

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 Speed
    float *Pi; //Best Position
    float XFit; //Fitness value
    float PFit; //Best position fitness value
} PARTICLE;

//Swarm definition
typedef struct {
    unsigned int numParticles; //Number of particles of the swarm
    unsigned int numParameters; //Number of parameters of the equation
    unsigned int idGbest; //id of the best particle
    float Vmin;
    float Vmax;
    float c1; //Random constant
    float c2; //Random constant
    PARTICLE *particle; //swarm
} SWARM;

const unsigned int NUMBER_PARTICLES = 80;
const unsigned int NUMBER_PARAMETERS = 2;
const unsigned int NUMBER_ITERATIONS = 200;
const float LimiteInf = 0;
const float LimiteSup = 10;
const float C1 = 2; //por teoria toman este valor
const float C2 = 2;
const float Vmin = -1.0;
const float Vmax = 1.0;

PARTICLE *CreateParticle(void);
SWARM *CreateSwarm(const unsigned int numParticles, const unsigned int numParameters);
void InitSwarm(SWARM *pSwarm, const float infL, const float supL, const float Vmin, const float Vmax, const float c1, const float c2);
void InitBest(SWARM *pSwarm);
void UpdateSpeed(SWARM *pSwarm);
void UpdatePosition(SWARM *pSwarm);
void UpdateBest(SWARM *pSwarm);
void EvaluateSwarm(SWARM *pSwarm);
void DeleteSwarm(SWARM *pSwarm);
void ShowParticle(SWARM *pSwarm, const unsigned int i);
void ShowSwarm(SWARM *pSwarm);

int main(void)
{
    SWARM *expSwarm;
    unsigned int it = 0;
    unsigned int maxIt = NUMBER_ITERATIONS;
    //Crear memoria para el enjambre
    expSwarm = CreateSwarm(NUMBER_PARTICLES, NUMBER_PARAMETERS);
```

```
//Inicializar posicones de las partículas ente los límite del espacio de búsqueda del problema
InitSwarm(expSwarm, LimiteInf, LimiteSup, Vmin, Vmax, C1, C2);
//Iniciar el fitness de cada partícula e identificar el fitness de la mejor global
InitBest(expSwarm);
```

```
while((!it < maxIt) && (50 - expSwarm -> particle[expSwarm -> idGbest].Pfit) > 0.00001)
{
    //Evaluar las partículas del enjambre en la ecuacion
    EvaluateSwarm(expSwarm);
    UpdateSpeed(expSwarm);
    UpdatePosition(expSwarm);
    UpdateBest(expSwarm);
    //Mostrar todas las partículas del enjambre
    ShowSwarm(expSwarm);
    printf("idGbest: %d \n", expSwarm -> idGbest);
    printf("it = %d -----\n", it);
    it++;
}
```

```
//Liberar la memoria del enjambre
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, const float Vmin, const float Vmax, const float c1, const float c2)
{
    unsigned int i, j;
    double r;
```

```

pSwarm -> c1 = c1;
pSwarm -> c2 = c2;

pSwarm -> Vmin = Vmin;
pSwarm -> Vmax = Vmax;

//Para todas las particulas
for(i = 0; i < pSwarm -> numParticles; i++)
{
    //Para todos los parametros
    for(j = 0; j < pSwarm -> numParameters; j++)
    {
        r = ((double) rand() / RAND_MAX) * (supL - infL) + infL; //Valor random entre InfL-SupL
        //Posicion aleatoria
        pSwarm -> particle[i].Xi[j] = r;
        //Asignamos velocidad 0, pues aun no se empiezan a mover
        pSwarm -> particle[i].Vi[j] = 0;
        //Peso aleatorio
        pSwarm -> particle[i].Pi[j] = r;
    }
    pSwarm -> particle[i].XFit = 0;
    pSwarm -> particle[i].PFit = 0;
}

return;
}

void InitBest(SWARM *pSwarm)
{
    unsigned int i;
    float best;
    best = 0;

    for(i = 0; i < pSwarm -> numParticles; i++)
    {
        //En la inicialización el mejor peso es directamente el peso de la particula, pues no hay mas en el historico
        pSwarm -> particle[i].PFit = pSwarm -> particle[i].XFit;
        if(pSwarm -> particle[i].XFit > best)
        {
            //Comparar los pesos de las particulas del ejambre poara saber cual de ellas es la mejor
            pSwarm -> idGbest = i;
            best = pSwarm -> particle[i].PFit;
        }
    }

    return;
}

void EvaluateSwarm(SWARM *pSwarm)
{
    //Evaluamos las particulas en la ecuacion
    unsigned int i;
    for(i = 0; i < pSwarm -> numParticles; i++)
    {
        //F(x,y) = (x-5)^2 + (y-5)^2
        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[1] - 5));
    }
    return;
}

```

```

void UpdateSpeed(SWARM *pSwarm)
{
    unsigned int i, j;
    float y1, y2, aux;

    srand(time(NULL));

    //Para todas las partículas
    for(i = 0; i < pSwarm -> numParticles; i++)
    {
        //para todos los parámetros
        for(j = 0; j < pSwarm -> numParameters; j++)
        {
            //Variables aleatorias
            y1 = ((double)rand())/RAND_MAX;
            y2 = ((double)rand())/RAND_MAX;
            //Cálculo de la formula que marca el algoritmo
            aux = pSwarm -> particle[i].Vi[j] + pSwarm -> c1 * y1 * (pSwarm -> particle[i].Pi[j] - pSwarm -> particle[i].Xi[j]) +
                pSwarm -> c2 * y2 * (pSwarm -> particle[pSwarm -> idGbest].Pi[j] - pSwarm -> particle[i].Xi[j]);
            if(aux > pSwarm -> Vmax)
            {
                pSwarm -> particle[i].Vi[j] = pSwarm -> Vmax;
                continue;
            }
            if(aux < pSwarm -> Vmin)
            {
                pSwarm -> particle[i].Vi[j] = pSwarm -> Vmin;
                continue;
            }

            pSwarm -> particle[i].Vi[j] = aux;
        }
    }

    return;
}

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

    //para todas las particulas
    for(i = 0; i < pSwarm -> numParticles; i++)
    {
        //para todos los parametros
        for(j = 0; j < pSwarm -> numParameters; j++)
        {
            //posicion actual mas la velocidad
            pSwarm -> particle[i].Xi[j] += pSwarm -> particle[i].Vi[j];
        }
    }

    return;
}

void UpdateBest(SWARM *pSwarm)
{
    unsigned int i, j;
    //Peso de la mejor partícula del enjambre
    float best = pSwarm -> particle[pSwarm -> idGbest].Pfit;

```

```

//Para todas las particulas
for(i = 0; i < pSwarm -> numParticles; i++)
{
    //Si el peso de actual es mayor que el del historico de esa particula
    if(pSwarm -> particle[i].XFit > pSwarm -> particle[i].PFit)
    {
        //Para todos los parametros
        for(j = 0; j < pSwarm -> numParameters; j++)
        {
            //Se actualizan los pesos optimos
            pSwarm -> particle[i].Pi[j] = pSwarm -> particle[i].Xi[j];
            pSwarm -> particle[i].PFit = pSwarm -> particle[i].XFit;
        }
    }
    //Si el mejor peso actual de la particula es mejor que el de la mejor del ejambre
    if(pSwarm -> particle[i].PFit > best)
    {
        //Se actualiza el id y peso del mejor
        pSwarm -> idGbest = i;
        best = pSwarm -> particle[i].PFit;
    }
}

return;
}

```

```

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

```

```

{
    unsigned int j;

    printf("\nX%d : ", i);
    for(j = 0; j < pSwarm -> numParameters; j++)
    {
        printf("%2.4f ", pSwarm -> particle[i].Xi[j]);
    }
    printf("\nV%d : ", i);
    for(j = 0; j < pSwarm -> numParameters; j++)
    {
        printf("%2.4f ", pSwarm -> particle[i].Vi[j]);
    }
    printf("\nP%d : ", i);
    for(j = 0; j < pSwarm -> numParameters; j++)
    {
        printf("%2.4f ", pSwarm -> particle[i].Pi[j]);
    }
    printf("\nXFit%d : ", i);
    printf("%2.4f ", pSwarm -> particle[i].XFit);

    printf("\nPFit%d : ", i);
    printf("%2.4f\n", pSwarm -> particle[i].PFit);

    return;
}

```

```

void ShowSwarm(SWARM *pSwarm)

```

```

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

```

```
    return;
}

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;
}
```


Pruebas y resultados

<pre>X0 : 8.4019 3.9438 V0 : 0.0000 0.0000 P0 : 8.4019 3.9438 XFit0 : 37.3117 PFit0 : 37.3117 X1 : 8.9034 0.5912 V1 : 1.0724 -7.3932 P1 : 8.9034 0.5912 XFit1 : 33.0788 PFit1 : 33.0788 X2 : 8.8869 3.0892 V2 : -0.2296 1.1137 P2 : 8.8869 3.0892 XFit2 : 23.9071 PFit2 : 23.9071 X3 : 3.9867 7.0699 V3 : 0.6345 -0.6124 P3 : 3.9867 7.0699 XFit3 : 40.0901 PFit3 : 40.0901 X4 : 7.4515 3.0726 V4 : 4.6737 -2.4671 P4 : 7.4515 3.0726 XFit4 : 44.7703 PFit4 : 44.7703 ldGbest: 4 it = 0 -----</pre>	<pre>X0 : 7.4890 3.4203 V0 : -0.9128 -0.5236 P0 : 8.4019 3.9438 XFit0 : 37.3117 PFit0 : 37.3117 X1 : 7.2483 -2.2617 V1 : -1.6552 -2.8528 P1 : 8.9034 0.5912 XFit1 : 15.3254 PFit1 : 33.0788 X2 : 8.1962 4.1936 V2 : -0.6907 1.1043 P2 : 8.1962 4.1936 XFit2 : 31.2409 PFit2 : 31.2409 X3 : 5.0565 5.8028 V3 : 1.0698 -1.2671 P3 : 5.0565 5.8028 XFit3 : 44.6886 PFit3 : 44.6886 X4 : 12.1252 0.6055 V4 : 4.6737 -2.4671 P4 : 7.4515 3.0726 XFit4 : 40.2754 PFit4 : 44.7703 ldGbest: 4 it = 1 -----</pre>	<pre>X0 : 6.7857 2.7174 V0 : -0.7033 -0.7028 P0 : 6.7857 2.7174 XFit0 : 41.3091 PFit0 : 41.3091 X1 : 7.4022 8.7196 V1 : 0.1539 10.9813 P1 : 8.9034 0.5912 XFit1 : -7.7868 PFit1 : 33.0788 X2 : 7.2663 4.6636 V2 : -0.9299 0.4700 P2 : 7.2663 4.6636 XFit2 : 39.1338 PFit2 : 39.1338 X3 : 6.4272 4.0884 V3 : 1.3707 -1.7144 P3 : 6.4272 4.0884 XFit3 : 49.3523 PFit3 : 49.3523 X4 : 9.3752 3.3469 V4 : -2.7500 2.7414 P4 : 7.4515 3.0726 XFit4 : -20.0800 PFit4 : 44.7703 ldGbest: 3 it = 2 -----</pre>	<pre>X0 : 5.7381 2.8385 V0 : -1.0477 0.1211 P0 : 5.7381 2.8385 XFit0 : 41.6011 PFit0 : 41.6011 X1 : 7.0192 -0.3802 V1 : -0.3830 -9.0998 P1 : 8.9034 0.5912 XFit1 : 30.3940 PFit1 : 33.0788 X2 : 6.0668 4.8082 V2 : -1.1995 0.1446 P2 : 6.0668 4.8082 XFit2 : 44.7508 PFit2 : 44.7508 X3 : 7.7979 2.3741 V3 : 1.3707 -1.7144 P3 : 6.4272 4.0884 XFit3 : 47.1322 PFit3 : 49.3523 X4 : 2.7184 7.0797 V4 : -6.6568 3.7328 P4 : 7.4515 3.0726 XFit4 : 28.1250 PFit4 : 44.7703 ldGbest: 3 it = 3 -----</pre>	<pre>X0 : 5.3523 3.7107 V0 : -0.3858 0.8722 P0 : 5.3523 3.7107 XFit0 : 44.7833 PFit0 : 44.7833 X1 : 7.1490 0.0835 V1 : 0.1299 0.4636 P1 : 8.9034 0.5912 XFit1 : 16.9769 PFit1 : 33.0788 X2 : 4.9831 4.5455 V2 : -1.0837 -0.2627 P2 : 4.9831 4.5455 XFit2 : 48.8252 PFit2 : 48.8252 X3 : 7.2524 1.7035 V3 : -0.5455 -0.6706 P3 : 6.4272 4.0884 XFit3 : 35.2764 PFit3 : 49.3523 X4 : 2.7284 3.9233 V4 : 0.0101 -3.1564 P4 : 7.4515 3.0726 XFit4 : 40.4692 PFit4 : 44.7703 ldGbest: 3 it = 4 -----</pre>
<pre>X0 : 5.9989 4.8099 V0 : 0.6466 1.0992 P0 : 5.9989 4.8099 XFit0 : 48.2137 PFit0 : 48.2137 X1 : 7.4358 8.6001 V1 : 0.2868 8.5167 P1 : 8.9034 0.5912 XFit1 : 21.2095 PFit1 : 33.0788 X2 : 4.3633 4.0242 V2 : -0.6198 -0.5213 P2 : 4.3633 4.0242 XFit2 : 49.7932 PFit2 : 49.7932 X3 : 5.5533 2.4848 V3 : -1.6991 0.7814 P3 : 6.4272 4.0884 XFit3 : 34.0593 PFit3 : 49.3523 X4 : 9.3894 0.5414 V4 : 6.6609 -3.3819 P4 : 7.4515 3.0726 XFit4 : 43.6807 PFit4 : 44.7703 ldGbest: 2 it = 5 -----</pre>	<pre>X0 : 5.0746 5.4370 V0 : -0.9243 0.6270 P0 : 5.0746 5.4370 XFit0 : 48.9661 PFit0 : 48.9661 X1 : 3.2163 -2.6925 V1 : -4.2195 -11.2927 P1 : 8.9034 0.5912 XFit1 : 31.1058 PFit1 : 33.0788 X2 : 3.7435 3.5030 V2 : -0.6198 -0.5213 P2 : 4.3633 4.0242 XFit2 : 48.6425 PFit2 : 49.7932 X3 : 4.8165 4.2320 V3 : -0.7368 1.7472 P3 : 6.4272 4.0884 XFit3 : 43.3677 PFit3 : 49.3523 X4 : 10.4058 3.9744 V4 : 1.0165 3.4331 P4 : 7.4515 3.0726 XFit4 : 10.8543 PFit4 : 44.7703 ldGbest: 2 it = 6 -----</pre>	<pre>X0 : 3.4671 5.2150 V0 : -1.6075 -0.2220 P0 : 3.4671 5.2150 XFit0 : 49.8035 PFit0 : 49.8035 X1 : 6.0558 2.9938 V1 : 2.8395 5.6863 P1 : 8.9034 0.5912 XFit1 : -12.3566 PFit1 : 33.0788 X2 : 3.6266 3.6979 V2 : -0.1169 0.1949 P2 : 4.3633 4.0242 XFit2 : 46.1800 PFit2 : 49.7932 X3 : 6.0720 5.8812 V3 : 1.2555 1.6493 P3 : 6.0720 5.8812 XFit3 : 49.3765 PFit3 : 49.3765 X4 : 4.1633 6.9747 V4 : -6.2426 3.0003 P4 : 7.4515 3.0726 XFit4 : 19.7254 PFit4 : 44.7703 ldGbest: 0 it = 7 -----</pre>	<pre>X0 : 1.8596 4.9931 V0 : -1.6075 -0.2220 P0 : 3.4671 5.2150 XFit0 : 47.6039 PFit0 : 49.8035 X1 : 6.4880 9.3135 V1 : 0.4322 6.3197 P1 : 6.4880 9.3135 XFit1 : 44.8605 PFit1 : 44.8605 X2 : 3.8195 5.0150 V2 : 0.1930 1.3171 P2 : 4.3633 4.0242 XFit2 : 46.4181 PFit2 : 49.7932 X3 : 7.0002 7.4213 V3 : 0.9282 1.5401 P3 : 6.0720 5.8812 XFit3 : 48.0742 PFit3 : 49.3765 X4 : -0.1674 5.0490 V4 : -4.3307 -1.9257 P4 : -0.1674 5.0490 XFit4 : 45.4004 PFit4 : 45.4004 ldGbest: 0 it = 8 -----</pre>	<pre>X0 : 2.2285 4.9171 V0 : 0.3690 -0.0760 P0 : 3.4671 5.2150 XFit0 : 40.1377 PFit0 : 49.8035 X1 : 1.2452 8.1340 V1 : -5.2427 -1.1795 P1 : 6.4880 9.3135 XFit1 : 29.1798 PFit1 : 44.8605 X2 : 4.1658 5.6447 V2 : 0.3463 0.6297 P2 : 4.3633 4.0242 XFit2 : 48.6063 PFit2 : 49.7932 X3 : 6.3036 7.9147 V3 : -0.6967 0.4934 P3 : 6.0720 5.8812 XFit3 : 40.1362 PFit3 : 49.3765 X4 : -1.4778 3.3799 V4 : -1.3104 -1.6691 P4 : -0.1674 5.0490 XFit4 : 23.2956 PFit4 : 45.4004 ldGbest: 0 it = 9 -----</pre>

Las primeras pruebas fueron cambiando el número de partículas y viendo cómo este cambio afectaba el peso de la solución encontrada. La conclusión fue que **A MAYOR NÚMERO DE PARTÍCULAS MEJORA EL RESULTADO**. Este número depende de la aplicación y se puede determinar de forma experimental, aunque los resultados generales marcan un rango de 20 a 80 partículas por enjambre en la mayoría de los problemas.

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-100.0	100.0	49.8035

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
20	10	0	10	-100.0	100.0	49.9074

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
50	10	0	10	-100.0	100.0	49.9760

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
100	10	0	10	-100.0	100.0	49.9865

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
200	10	0	10	-100.0	100.0	49.9982

Se experimentó también cambiando el rango de búsqueda de la solución. La conclusión aquí fue que **UN RANGO PEQUEÑO Y NO MUY ALEJADO DE LA SOLUCIÓN FUNCIONA MEJOR**. La desventaja de acotar el rango de búsqueda es que no siempre se conoce la solución, entonces existe la posibilidad de que dejemos a la solución fuera del rango de búsqueda, por lo que este parámetro debe ser utilizado con cuidado.

Otra cosa que se pudo observar es que las partículas se perdían con un rango muy grande y terminaban muy lejos de la solución o incluso no se podían encontrar.

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-100.0	100.0	49.8139

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	-10	10	-100.0	100.0	49.7167

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	-20	20	-100.0	100.0	49.1216

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	-50	50	-100.0	100.0	Indef

También se experimentó con el número de iteraciones y se pudo observar que **NO SIEMPRE MÁS ITERACIONES SIGNIFICA ESTAR MÁS CERCA DE LA SOLUCIÓN**. Para mi caso en particular 100 iteraciones fue lo que más me acercó a la solución.

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-100.0	100.0	49.9731

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	20	0	10	-100.0	100.0	49.0602

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	50	0	10	-100.0	100.0	49.3008

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	100	0	10	-100.0	100.0	49.9991

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	200	0	10	-100.0	100.0	49.9823

La velocidad es un parámetro que usualmente se controla, pues después de cierto tipo de iteraciones las partículas suelen explorar y alejarse demasiado de la solución. Lo que pude concluir es que **LIMITAR LA VELOCIDAD AYUDA A LLEGAR A LA SOLUCIÓN DE UNA MANERA MÁS RÁPIDA.**

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-100.0	100.0	49.2789

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-50.0	50.0	49.9830

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-20.0	20.0	49.9889

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-5.0	5.0	49.9483

Particles	Iterations	LímiteInferior	LímiteSuperior	Vmin	Vmax	BEST
5	10	0	10	-1.0	1.0	49.9722

Para ahorrar algunas iteraciones se pone el parámetro de error, que sirve para que cuando encontremos una solución que cumpla el error la ejecución se detenga ahorrando tiempo de cómputo si el programa lo encuentra antes de agotar el límite de iteraciones.

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
5	100	1	0	10	-1.0	1.0	2

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
5	100	0.5	0	10	-1.0	1.0	17

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
5	100	0.1	0	10	-1.0	1.0	4

Particles	Iterations	Error	LímiteInf	LímiteSup	Vmin	Vmax	Iteration
5	100	0.001	0	10	-1.0	1.0	99

Particles	Iterations	Error	LímiteInf	LímiteSup	Vmin	Vmax	Iteration
5	100	0.0001	0	10	-1.0	1.0	99

Las últimas pruebas se realizaron utilizando los parámetros que resultaron óptimos para mi implementación del algoritmo y registrando en qué iteración alcanza una solución con cierto margen de error. Observando los resultados se puede observar que llega de una manera mucho más rápida a la solución, ahorrando iteraciones y por tanto tiempo de cómputo.

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
200	100	1	0	10	-1.0	1.0	0

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
200	100	0.5	0	10	-1.0	1.0	1

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
200	100	0.001	0	10	-1.0	1.0	16

Particles	Iterations	Error	LímiteInferior	LímiteSuperior	Vmin	Vmax	Iteration
200	100	0.0001	0	10	-1.0	1.0	65

Conclusión

Incluso después de haber implementado el algoritmo se deben definir ciertos parámetros o constantes para hacer que tenga el mejor rendimiento. Estos parámetros pueden definirse de forma experimental, pero de igual modo ya hay cifras determinadas que suelen funcionar para la mayor parte de las aplicaciones. De cualquier modo es bueno revisar cómo influyen estos parámetros en nuestra implementación particular, pues la eficiencia depende de muchos factores muy particulares (SO, hardware, frecuencias, ciclos, etc), que hacen que ciertos números funcionen mejor que lo que la teoría dicta.

Siempre es bueno experimentar un poco con los algoritmos antes de iniciar a aplicarlos, pues podemos entenderlos mejor y saber qué ajustes hacerle para que funcione de una manera mejor, más eficiente y rápida.