UNIVERSIDAD DE GUANAJUATO

DIVISIÓN DE INGENIERÍAS

CAMPUS – IRAPUATO SALAMANCA

Inteligencia Artificial

PROF. GARCÍA CAPULÍN CARLOS HUGO

TAREA 5

Implementación PSO

Martínez Lona Verónica Montserrat

Viernes 3 Marzo, 2017

Introducción:

Se terminó la implementación del método POS para la resolución de una ecuación particular. Las funciones que se dearrollaron fueron la actualización de la velocidad, posición, mejor partícula y una función de inicialización.

Desarrollo:

Los pasos del algoritmo quedaron finalmente de la siguiente manera:

1. Crear la memoria para el enjambre.
2. Inicializar las posiciones de las partículas entre los límites del espacio de búsqueda.
3. Inicializar el fitness de cada partícula y el fitness global.
4. Evaluar las partículas del enjambre en la ecuación.
5. Actualizar la velocidad.
6. Actualizar la posición.
7. Actualizar la mejor partícula.
8. Mostrar el enjambre.
9. Mostrar el id de la mejor partícula histórica.
10. Liberar la memoria del enjambre.

Las funciones nuevas que se definierons fueron:

- *void InitBest(SWARM *pSwarm);*
  - Inicializamos las variables de fitness y mejor fitness para la primera iteración.
- *void UpdateSpeed(SWARM *pSwarm);*
  - Actualizamos el valor de la velocidad con la fórmula definida por el algoritmo.
- *Void UpdatePosition(SWARM *pSwarm);*
  - Actualizamos el valor de la posición (Posición actual + velocidad).
- *void UpdateBest(SWARM *pSwarm);*
  - Hacemos las comparaciones pertinentes y actualizamos el fitness óptimo y el índice de la mejor partícula histórica global si es necesario.

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 c1; //Random constant
  float c2; //Random constant
  PARTICLE *particle; //swarm
} SWARM;

const unsigned int NUMBER_PARTICLES = 5;
const unsigned int NUMBER_PARAMETERS = 2;
const unsigned int NUMBER_ITERATIONS = 5;
const float LimiteInf = 0;
const float LimiteSup = 10;
const float C1 = 2; //por teoria toman este valor
const float C2 = 2;

PARTICLE *CreateParticle(void);
SWARM *CreateSwarm(const unsigned int numParticles, const unsigned int numParameters);
void InitSwarm(SWARM *pSwarm, const float LInf, const float LSup, 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, C1, C2);
  //Iniciar el fitness de cada partícula e identificar el fitness de la mejor global
  InitBest(expSwarm);
```

```
 while(it < maxIt)
 {
  //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("-------------------------------------------------\n");
  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 c1, const float c2)
{
 unsigned int i, j;
 double r;

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

```
 //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 inicalización 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)ª2
  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;

 //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);
   //Càlculo de la formula que marca el algoritmo
   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]);
  }
 }

 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;
  }
```

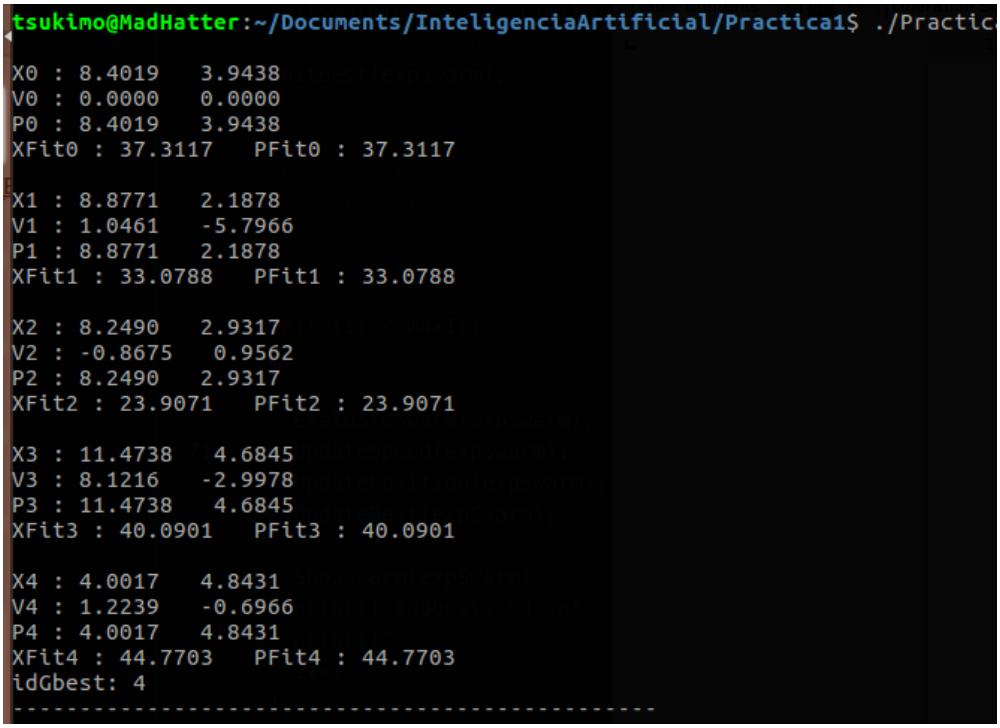
```
 }

 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("PFit%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

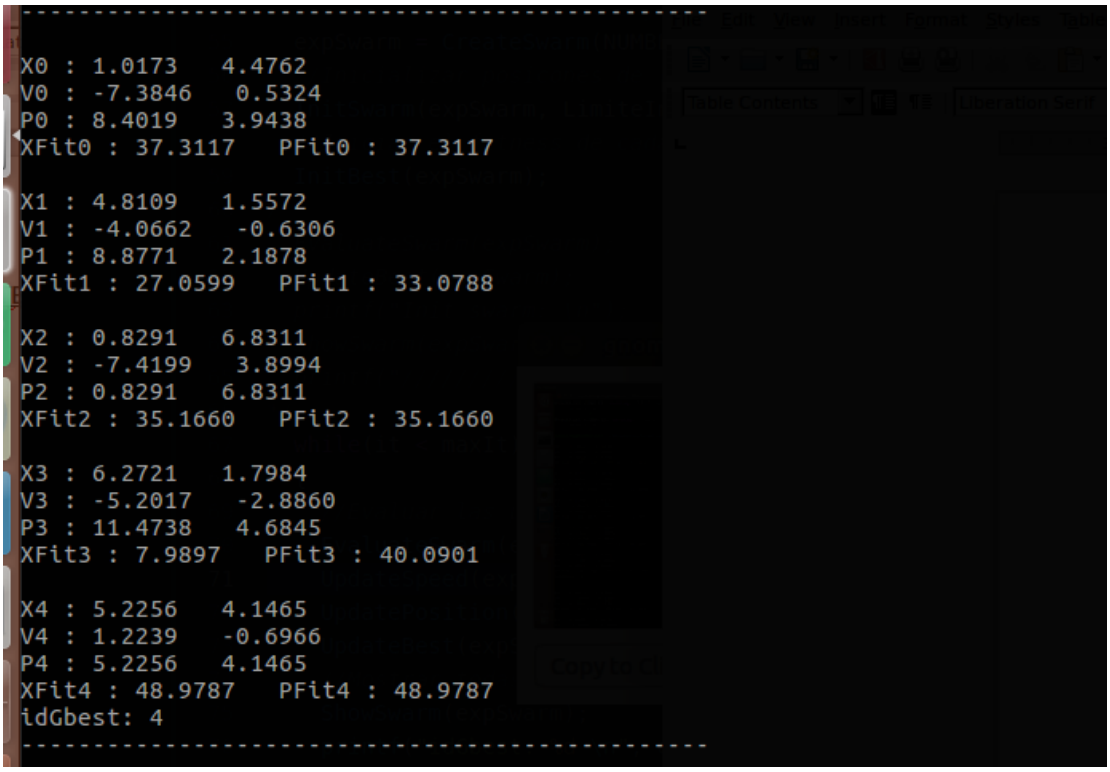
El programa hacer todo de manera automática , pues la ecuación está definida en el código. Por lo que no es necesario ingresar ningún parámetro o argumento. Las pruebas consistieron en revisar que se estuvieran actualizando correctamente los pesos de las partículas y el valor del fitness óptimo.

Si revisamos la tabla se puede ver como se van actualizando los valores tanto como posición, velocidad, el fitness de la partícula en ese momento y de darse el caso se actualiza el mejor fitness histórico y la id de la mejor partícula global. Los valores cambian de una forma psuedo aletoria, pues a pesar de estar guiados por una ecuación hay variables aleatorias (y1, y2) que afectan su movimiento, pero no son tan relevantes como para hacer cambios radicales, por lo que usualmente las partículas no desvían su movimiento de una manera brusca, si no que lo hacen paulatinamente (y en ocasiones regresan a una posición más cercana a la solución).

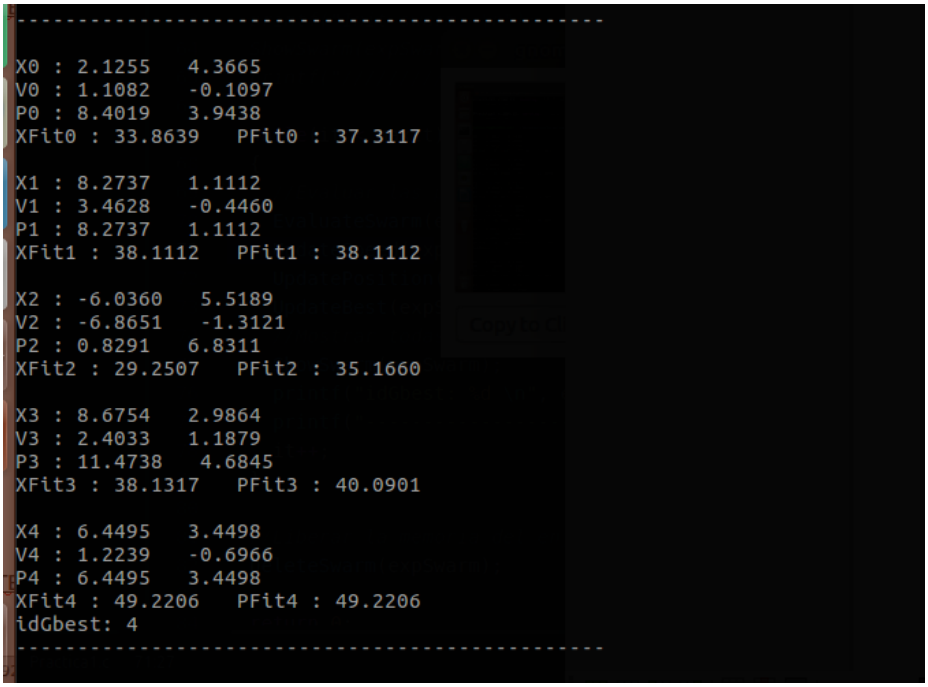
- ITERACIÓN 1



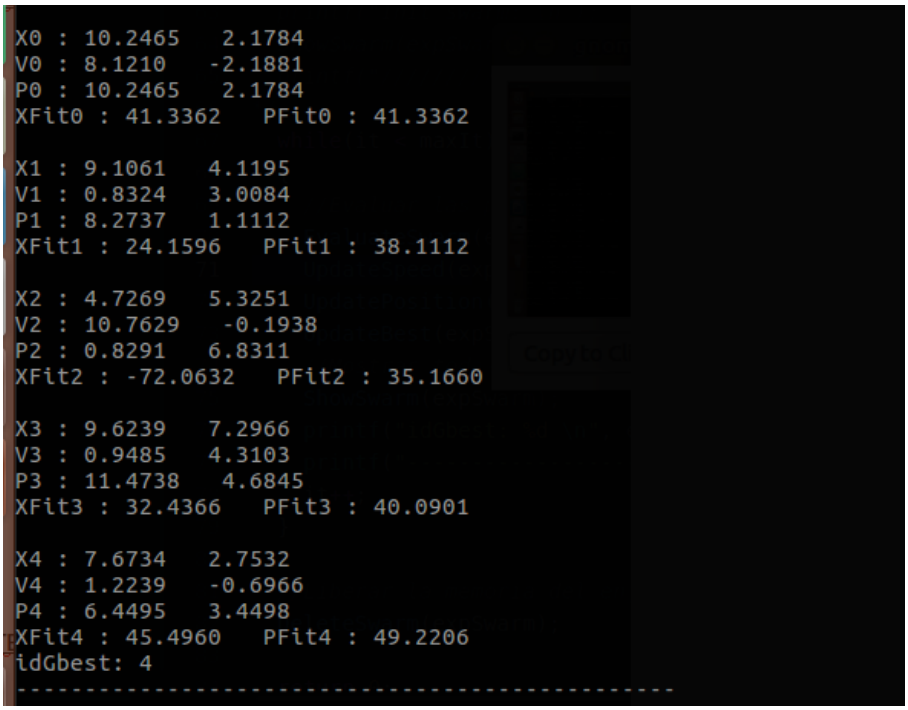
- ITERACIÓN 2



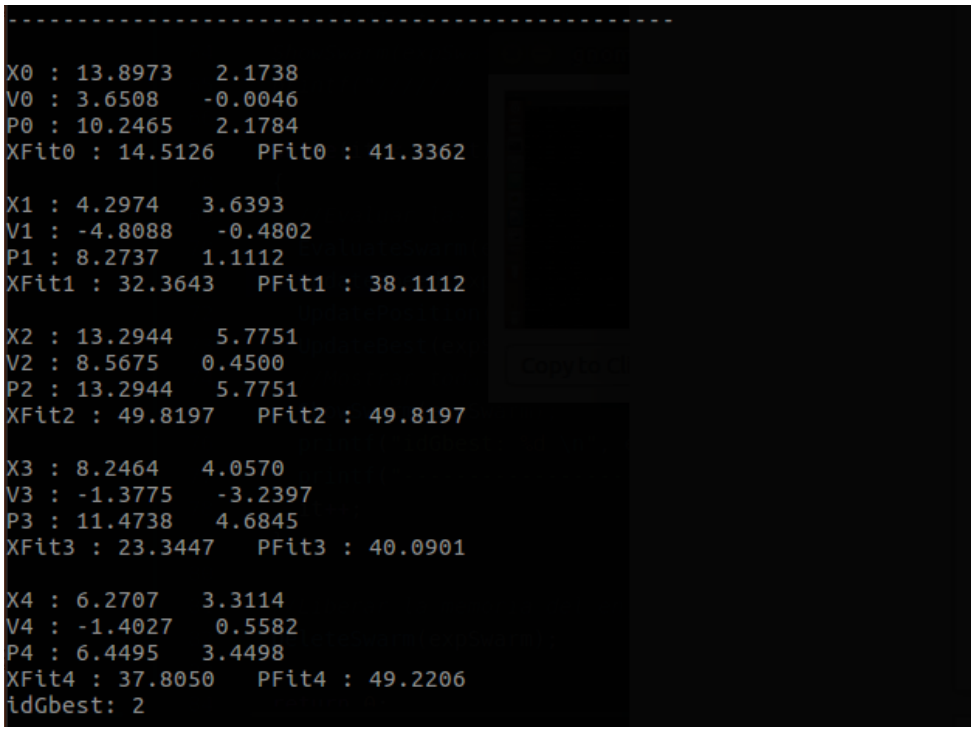
- ITERACIÓN 3



- ITERACIÓN 4



- ITERACIÓN 5



	It = 1	It = 2	It = 3	It = 4	It = 5
X0	8,4019 / 3,9438	1,0173 / 4,4762	2,1255 / 4,3665	10,2465 / 2,1784	13,8973 / 2,1738
V0	0,0000 / 0,0000	-7,3846 / 0,5324	1,1082 / -0,1097	8,1210 / -21881	3,6508 / -0.0046
P0	8,4019 / 3,9438	8,4019 / 3,9438	8,4019 / 3,9438	10,2465 / 2,1784	10,2465 / 2,1784
XFit0	37,3117	37,3117	33,8639	41,3362	14,5126
PFit0	37,3117	37,3117	37,3117	41,3362	41,3362
X1	8,8771 / 2,1878	4,8109 / 1,5572	8,2737 / 1,1112	9,1061 / 4,1195	4,2974 / 3,6393
V1	1,0461 / -5,7966	-4,0662 / -0,6306	2,4628 / -0,4460	0,8324 / 3,0084	-4,8088 / -0,4802
P1	8,8771 / 2,1878	8,771 / 2,1878	8,2737 / 1,1112	8,2737 / 1,1112	8,2737 / 1,1112
XFit1	33,0788	27,0599	38,1112	24,1596	32,3643
PFit1	33,0788	33,0788	38,1112	38,1112	38,1112
X2	8,2490 / 2,9317	0,8291 / 6,8311	-6,0360 / 5,5189	4,7269 / 5,3251	13,2944 / 5,7751
V2	-0,8675 / 0,9562	-7,4199 / 3,8994	-6,8651 / -1,3121	10,7629 / -0,1938	8,5675 / 0,4500
P2	8,2490 / 2,9317	0,8291 / 6,8311	0,8291 / 6,8311	0,8291 / 6,8311	13,2944 / 5,7751
XFit2	23,9071	35,1660	29,2507	-72,0632	49,8197
PFit2	23,9071	35,1660	35,1660	35,1660	49,8197
X3	11,4738 / 4,6845	6,2721 / 1,7984	8,6754 / 2,9864	9,6239 / 7,2966	8,2464 / 4,0570
V3	8,1216 / -2,9978	-5,2017 / -2,8860	2,4033 / 1,1879	0,9485 / 4,3103	-1,3775 / -3,2397
P3	11,4738 / 4,6845	11,4738 / 4,6845	11,4738 / 4,6845	11,4738 / 4,6845	11,4738 / 4,6845
XFit3	40,0901	7,9897	38,1317	32,4366	23,3447
PFit3	40,0901	40,0901	40,0901	40,0901	40,0901
X4	4,0017 / 4,8431	5,2256 / 4,1465	6,4495 / 3,4498	7,6734 / 2,7532	6,2707 / 3,4498
V4	1,2239 / -0,6966	1,2239 / -0,6966	1,2239 / -0,6966	1,2239 / -0,6966	-1,4027 / 0,5582
P4	4,0017 / 4,8431	5,2256 / 4,1465	6,4495 / 3,4498	6,4495 / 3,4498	6,4495 / 3,4498
XFit4	44,7703	48,9787	49,2206	45,4960	37,8050
PFit4	44,7703	48,9787	49,2206	49,2206	49,2206
idBest	4	4	4	4	2

Conclusión

El método POS pude llegar a parecer algo complicado en la teoría, pero su implementación en código resultó bastante más simple, aunque algo confuso si no se estructura de una manera adecuada y claro, se utilizan nombres descriptivos, pues se utilizan varias estructuras anidadas, lo que puede causar confusiones.

A pesar de lo sencillo que pueda ser en código el algoritmo es muy poderoso y en pocas iteraciones puede llegar a algo bastante cercano, incluso estando las posiciones afectadas por números aleatorios. Durante la prueba del algoritmo tuve bastantes problemas, pues no estaba prestando atención al orden en el que se realizaban las funciones, cosa que es fundamental en en la obtención correcta de los datos.

En esta parte sólo hemos hecho una pequeña aplicación del algoritmo, pero que, si se constrasta el tiempo de desarrollo y tiempo de cómputo con algún otro método de resolución de ecuaciones de dos variables, se puede observar la gran diferencia en la facilidad de implementación y la velocidad con la que se obtiene un resultado aceptable.

Incluso si se hiciera con otra estructura de control podría ser aún más veloz, por ejemplo listas enlazadas para hacerlo de una dimensión variable, y demás.