Representación de bolas n-dimensionales

Javier Pérez

Report LSI–91–36

Representation of N-Dimensional balls

Javier Pérez Campo

Abstract

This work presents a new method to represent N-Dimensional balls of points. The method translates the points situated on a N-Dimensional space to a space with less dimensions. The objective is to mantain the distances between points in the target space as close as possible to the distances between points in the original space.

The target space will normally have two or three dimensions. In the case of two dimensions the points will be shown on a black and white screen or on paper. In the case of three dimensions the points can be shown on a color screen, using the color of the points as the third dimension.

REPRESENTACION DE BOLAS N-DIMENSIONALES

JAVIER PEREZ CAMPO

INDICE

1 OBJETIVO DEL TRABAJO

El objeto de este trabajo es diseñar un método de representación de bolas de puntos n-dimensionales. El método que se seguirá es el de trasladar los puntos que forman estas bolas desde su espacio original de **P** dimensiones hacia un espacio de **M** dimensiones intentando que se conserven las distancias entre los puntos. Una vez en el espacio de **M** dimensiones estos puntos serán visualizados.

El motivo por el que se realiza es el de proveer de un interfaz gráfico al sistema de adquisición automática de conocimientos **LINNEO** [MART91]. Este sistema permite en una de sus fases la agrupación de un conjunto de objetos en clases. Estos objetos serán los que deberán visualizarse así como las clases que forman.

2 METODO DE APROXIMACION

El dato que se utilizará es la distancia entre puntos. Se ajustará esta distancia según la distancia euclidiana. Se trata de colocar los puntos de modo que entre ellos se conserve la distancia. Esto salvo en casos muy particulares no es posible pues estamos pasando a un espacio en el que, normalmente, se tendrán menos dimensiones y por tanto menos libertad para colocar los puntos.

En consecuencia se colocarán los puntos tratando que el error cometido en la representación sea el mínimo.

Sea **N** el número de puntos, d_{ij} la distancia en el espacio **P**-dimensional entre los puntos i y j y r_{ij} la distancia en el espacio **M**-dimensional entre los puntos i y j, definimos el error como:

$$E = \sum_{j=1}^{N} \sum_{i=j+1}^{N} \left(r_{ij}^{\,2} - d_{ij}^{\,2} \right)^2$$

La distancia euclidiana tiene la expresión:

$$r_{ij} = \sqrt{\sum_{l=1}^{M} \left(c_{il} - c_{jl} \right)^2}$$

Siendo c_{il} la coordenada del punto i en la dimensión l.

Es importante que se puedan colocar (**M** + 1) puntos a una distancia exacta si se toman los puntos de la siguiente forma:

punto 0: 0, 0, 0, ..., 0, 0
punto 1: d, 0, 0, ..., 0, 0
.
.
.
punto M-1: d, e, f, ..., u, 0
punto M : d, e, f, ..., u, v

El punto **M** + 2 se colocará teniendo en cuenta su distancia respecto a los **M** + 1 primeros puntos. Como que en general no será posible ubicar este punto de modo que esté a las distancias requerida respecto de los **M** + 1 primeros puntos, se procurará colocarlo de modo que se minimicen las diferencias entre las distancias pedidas y las distancias a las que se sitúa el punto.

Los siguientes puntos además de tener en cuenta las distancias respecto a los **M** + 1 primeros puntos deberán tener en cuenta las distancias respecto a los puntos que ya han sido colocados, en concreto de punto **M** + 3 deberá tener en cuenta su distancia respecto al punto **M** + 2.

2.1 COLOCACION DE LOS PRIMEROS PUNTOS

Sabiendo que el primer punto es el 0,0, ..., 0. Por cada punto que se coloque se perderá un grado de libertad. Por tanto se fijarán las primeras coordenadas del punto, que serán diferentes de cero. Las coordenadas diferentes de cero serán tantas como tuviera el punto anterior más una.

En general y teniendo en cuenta las coordenadas a cero se cumple:

$$\text{Si } k < j,\ D_{jk} = \sqrt{\sum_{i=1}^{k} (c_{ki} - c_{ji})^2}$$

El punto 1 es muy fácil de colocar: $c_{11} = d_{10}$.

Explicitando la fórmula anterior se obtiene:

$$D_{0k}^2 = c_{k1}^2 + \quad ... \quad + c_{kk}^2$$

$$D_{1k}^2 = (c_{k1} - c_{11})^2 + c_{k2}^2 \quad ... + c_{kk}^2$$

.
.
.

$$D_{k-1k}^2 = (c_{k1} - c_{k-11})^2 + ... (c_{kk-1} - c_{k-1k-1})^2 + c_{kk}^2$$

Restando la primera ecuación del resto se obtiene:

$$D_{1k}^2 - D_{0k}^2 = (c_{k1} - c_{11})^2 - c_{k1}^2$$

$$D_{2k}^2 - D_{0k}^2 = (c_{k1} - c_{21})^2 + (c_{k2} - c_{22})^2 - c_{k1}^2 - c_{k2}^2$$

.
.
.

$$D_{k-1k}^2 - D_{0k}^2 = (c_{k1} - c_{k-11})^2 + ... + (c_{kk-1} - c_{k-1k-1})^2 - c_{k1}^2 ... - c_{kk-1}^2$$

Es por tanto un sistema triangular que es fácil de resolver. Si se simplifican los binomios al cuadrado se obtiene:

$$c_{kj} = \frac{D_{0k}^2 - D_{jk}^2 + c_{jj}^2 + \sum_{i=1}^{j-1} \left(c_{ji}^2 - 2 c_{ji} c_{ki} \right)}{2 c_{jj}}$$

Si j está entre 1 y k - 1.

Si j es igual a k se obtiene:

$$c_{kk} = \sqrt{D_{0k}^2 - \sum_{j=1}^{k-1} c_{kj}^2}$$

Esto último se desprende de la ecuación que se resta en el sistema de ecuaciones.

Como puede observarse c_{kk} tiene dos posibles soluciones: positiva y negativa. El error cometido al colocar los puntos no variará si se toma una u otra solución. Por tanto se escoge la solución positiva.

Puede observarse que todo esto es válido si c_{kk} es diferente de cero, pues si es cero el siguiente punto no podrá hallarse. Además el argumento de la raíz cuadrada

debe ser positivo pues sino este punto no existiría en un espacio de números reales. Se puede demostrar que si partimos de unas distancias que efectivamente correspondan a unos puntos en un espacio **P**-dimensional el argumento de la raíz será siempre mayor o igual a cero. Sin embargo esto no puede asegurarse si se utilizan distancias ficticias.

Para demostrarlo se verá que la interpretación geométrica del método de resolución expuesto corresponde a realizar una serie de traslaciones y rotaciones sobre los puntos originales que dan lugar los puntos hallados.

Sean inicialmente los puntos:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 0 & \dots & 0 \\ a'_{21} & a'_{22} & \dots & a'_{2n} \\ \dots & & & \\ a'_{n1} & a'_{n2} & \dots & a'_{nn} \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 0 & \dots & 0 \\ a''_{21} & 0 & \dots & 0 \\ \dots & & & \\ a''_{n1} & a''_{n2} & \dots & a_{nn} \end{pmatrix} \rightarrow \dots$$

$$\dots \rightarrow \begin{pmatrix} 0 & 0 & \dots & 0 & 0 \\ a^{n}_{21} & 0 & \dots & 0 & 0 \\ \dots & & & & \\ a^{n}_{n1} & a^{n}_{n2} & \dots & a^{n}_{n-1n} & 0 \end{pmatrix}$$

Las flechas indican como queda la matriz tras la aplicación de rotaciones y traslaciones.

Para pasar a la primera matriz realizamos una serie de traslaciones sobre todos los ejes para que el punto inicial vaya a parar al origen de coordenadas, el resto de transformaciones son giros de angulos trazados sobre una serie de hiperesferas.

La ecuación general de una hiperesfera de **n** dimensiones es:

$$(x_1 - a_1)^2 + (x_2 - a_2)^2 + \dots + (x_n - a_n)^2 = r^2$$

Siendo $\mathbf{a_i}$ el punto sobre el que está centrada y **r** su radio

Esta hiperesfera se toma centrada sobre el primer punto que deseamos colocar, que es el origen de coordenadas. Para hallar el siguiente punto se realiza un giro hacia el punto de la hiperesfera que cumple que todas sus componentes son cero excepto la componente de la primera dimensión. Esto es posible pues se tiene una ecuación con **n** grados de libertad.

Esto se repite sucesivamente con cada punto obtenido, sólo que para cada punto se utiliza una hiperesfera centrada en dicho punto. El giro que se da es un giro sobre la hiperesfera resultante de la intersección de las anteriores hiperesferas que pierde tantos grados de libertad como hiperesferas se intersectan menos uno. Así se va obteniendo el sistema de ecuaciones antes descrito.

Queda claro que la parte derecha del sistema es idéntica pero no así la parte izquierda. Dado que las transformaciones que realizamos: traslaciones y giros mantienen la distancia entre los puntos, las distancias originales, que son las que se utilizan en la parte izquierda del sistema son las mismas que existen tras realizar estos giros.

Por tanto es posible obtener las posiciones de **M** + 1 puntos en un espacio **M** dimensional que mantengan las distancias que originalmente tenían.

Puede aún darse el caso de que c_{kk} sea cero. Si se halla este caso significa que este punto no necesita una coordenada en esta última dimensión para cumplir las restricciones, por lo que se sustituye este punto por otro. El punto sustituido se guarda aparte pues ya se ha conseguido una buena ubicación. Estos puntos no hará falta que se hallen en la siguiente fase del algoritmo.

2.2 COLOCACION DEL RESTO DE PUNTOS

Para colocar el resto de puntos se utilizará un método de aproximación:

- Se dividirá la distancia entre dos puntos entre las **M** dimensiones de un modo uniforme o no (ya se verá como se realiza esta división). Una vez centrados sobre una dimensión se decidirá la coordenada de un punto en dicha dimensión en función de las coordenadas de dicha dimensión de los puntos ya colocados y la fracción de la distancia que pertenece a dicha dimensión.

En cada punto ya colocado tenemos una distancia a la que deberíamos colocar el nuevo punto pero existen dos problemas:

1. Muy probablemente no será posible satisfacer todas las distancias entre los puntos a colocar.

2. Dado que estamos en una recta está distancia puede tomarse tanto positiva como negativamente.

Para resolver el primer problema se toma la media de las posibles soluciones como solución final.

Para resolver el segundo problema se realizan los cálculos con ambas distancias y se determina en cada caso cuál es el signo (positivo o negativo) más conveniente para dicha distancia en función del error que se comete al tomar una u otra distancia.

Inicialmente se considera para todos el signo positivo, por lo que la primera aproximación es:

$$c_{kl}=\frac{\sum_{i=0}^{k-1}\left(c_{il}+D_{ik}\times P_l\right)}{k}$$

Siendo P_l la fracción de la distancia total para la dimensión l.

En sucesivas iteraciones se comprobará si el cambiar de signo una o varias distancias rebaja el error. Si así es se variará el signo y se realizarán iteraciones mientras el repaso a los signos haga bajar el error. Tras colocar la primera dimensión del punto se tendrá un error en dicha dimensión. Para tratar de compensar dicho error se pasará este error a la siguiente dimensión en la que la fórmula para hallar la aproximaxión varía a:

$$c_{kl}=\frac{\sum_{i=0}^{k-1}\left(c_{il}+D_{ik}\times P_l+E_i\right)}{k}$$

Donde el error E_i es:

$$E_i=-D_{ik}\times P_l\pm\sqrt{\left(D_{ik}\times P_l\right)^2+\left(D_{ik}\times P_{l-1}\right)^2-\left(c_{i\,l-1}-c_{k\,l-1}\right)^2}$$

Las dos soluciones indican las opciones que tenemos para tomar la distancia, es decir, en una u otra dirección.

Esto se deduce de imponer las condiciones que deseamos: que la distancia entre los puntos con las coordenadas que provocan el error sea la misma que con las coordenadas que no provocarían este error:

$$\sqrt{\sum_{j=1}^{M}(D_{ik}\times P_j)^2}=D_{ik}$$

$$\sqrt{\sum_{j=1}^{l-2}(D_{ik}\times P_j)^2+(c_{i\,l-1}-c_{k\,l-1})^2+(D_{ik}\times P_l+E_i)^2+\sum_{j=l+1}^{M}(D_{ik}\times P_j)^2}=D_{ik}$$

Donde **l** - 1 es la última dimensión que se ha colocado y tiene un error E_i que es lo que se ha de sumar en la dimensión **l** para compensar el error.

Si se igualan ambas ecuaciones y se simplifican, se obtiene:

$$(c_{i\,l-1}-c_{k\,l-1})^2+(D_{ik}\times P_l+E_i)^2=(D_{ik}\times P_{l-1})^2+(D_{ik}\times P_l)^2$$

Deshaciendo los cuadrados y simplificando se obtiene:

$$E_i^2+2E_i\times P_l\times D_{ik}+(c_{i\,l-1}-c_{k\,l-1})^2-(D_{ik}\times P_{l-1})^2=0$$

Resolviendo esta ecuación se obtiene el valor ya dado de E_i.

2.3 REPARTICION DE LA DISTANCIA

Hasta ahora se ha supuesto que existe un mecanismo mediante el cual a cada dimensión se le asignaba una parte proporcional de la distancia total, ahora se va a discutir este método.

Inicialmente se repartirá la distancia equitativamente entre todas las dimensiones, por tanto:

$$\underset{l=1}{\overset{M}{\forall}} d_{ijl}=\frac{D_{ij}}{\sqrt{M}}$$

Siendo d_{ijl} la distancia inicial para todas las dimensiones.

Esto se desprende de:

$$D_{ij}=\sqrt{\sum_{l=1}^{M}d_{ijl}^2}\;;\;D_{ij}^2=M\times d_{ijl}^2\;;\;d_{ijl}=\frac{D_{ij}}{\sqrt{M}}$$

Pero esta división puede no ser la mejor, es posible que algunas dimensiones tengan demasiado error, lo que las fuerze a colocar el punto más lejos de lo que sería óptimo, mientras que otras pueden tener demasiado poco lo que las fuerza a colocarlo demasiado cercano.

Por ello se realizará una redistribución de la distancia en función del error en cada dimensión.

Dado que el signo del error es significativo en este caso, Calcularemos el error con signo. Un signo negativo significa que la distancia entre los puntos es demasiado pequeña mientras que uno positivo significa que es demasiado grande.

Para proceder al cálculo se calcula el error introducido por cada punto, este error se eleva al cuadrado pero separando los errores positivos de los negativos. Cuando hemos calculado todos los errores se restan los positivos de los negativos y el resultado es el error de dicha dimensión. Se modificarán las proporciones de las distancias en las diversas dimensiones de modo que el error cometido en todas las dimensiones se iguale. El error que se tomará como referencia será cero.

Sea sig(i, j, l) el signo del error en la dimensión l del punto i con respecto al punto j, N el número de puntos y P_l la nueva proporción del error, el error con signo en la dimensión l es:

$$E_l = \sum_{j=1}^{N} \sum_{i=j+1}^{N} \left((c_{il} - c_{jl})^2 - d_{ij}^2 \times P_l \right)^2 \times \mathrm{sig}(i,j,l)$$

Dado que este error ha de ser igual a cero, se iguala este error a cero y se deshacen los cuadrados para despejar P_l.

$$0 = \sum_{j=1}^{N} \sum_{i=j+1}^{N} \left((c_{il} - c_{jl})^4 + d_{ij}^4 \times P_l^2 - 2(c_{il} - c_{jl})^2 \times d_{ij}^2 \times P_l \right) \times \mathrm{sig}(i,j,l);$$

Rompiendo los sumatorios:

$$P_l^2 \times \sum_{j=1}^{N} \sum_{i=j+1}^{N} d_{ij}^4 \times \mathrm{sig}(i,j,l) - 2P_l \times \sum_{j=1}^{N} \sum_{i=j+1}^{N} (c_{il} - c_{jl})^2 d_{ij}^2 \times \mathrm{sig}(i,j,l) +$$
$$+ \sum_{j=1}^{N} \sum_{i=j+1}^{N} (c_{il} - c_{jl})^4 \times \mathrm{sig}(i,j,l) = 0$$

Sea:

$$S_1 = \sum_{j=1}^{N} \sum_{i=j+1}^{N} (c_{il} - c_{jl})^2 \times d_{ij}^2 \times \mathrm{sig}(i,j,l)$$

$$S_2 = \sum_{j=1}^{N} \sum_{i=j+1}^{N} d_{ij}^4 \times \mathrm{sig}(i,j,l)$$

$$S_3 = \sum_{j=1}^{N} \sum_{i=j+1}^{N} (c_{il} - c_{jl})^4 \times \mathrm{sig}(i,j,l)$$

Se despeja P_l y se obtiene:

$$P_l = \frac{S_1 \pm \sqrt{S_1^2 - S_2 \times S_3}}{S_2}$$

Si el contenido de la raíz es menor que cero se toma el punto más cercano, o sea se iguala el contenido de la raíz a cero.

Existe el problema de escoger la raíz que más interesa, esto se hará en función del error. Si el error en la dimensión l es mayor que cero, la nueva proporción debe ser menor que la anterior. Si por el contrario el error es menor que cero, la proporción en esta dimensión debe subir.

La elección de la raíz depende de los valores de S_2 y E_l.

La relación entre el error y la anterior proporción P_l' es:

$E_l' = P_l'^2 S_2 - 2 P_l' S_1 + S_3$

Por definición de E_l', error con la proporción de error P_l'. Por tanto se cumple:

$$P_l' = \frac{S_1 \pm \sqrt{S_1^2 - S_2 \times (S_3 - E_l')}}{S_2}$$

Sean T+ y T- las soluciones positiva y negativa de P_l y T+' y T-' las soluciones positiva y negativa de P_l'.

Si S_2 es mayor que cero entonces:

Si $E_l' < 0$, entonces T+ > T+' y T- < T-' por tanto se tiene:

T- < T-' < T+' < T+

Dado que $P_l > P_l'$ se toma la solución T+.

Si $E_l' > 0$, entonces $T+ < T+'$ y $T- > T-'$ por tanto se tiene:

$$T-' < T- < T+ < T+'$$

Dado que $P_l < P_l'$ se toma la solución $T+$.

Por tanto siempre se toma la solución positiva.

Si S_2 es menor que cero entonces:

Si $E_l' < 0$, entonces $T+ < T+'$ y $T- > T-'$ por tanto se tiene:

$$T-' < T- < T+ < T+'$$

Dado que $P_l > P_l'$ se toma la solución $T-$.

Si $E_l' > 0$, entonces $T+ > T+'$ y $T- < T-'$ por tanto se tiene:

$$T- < T-' < T+' < T+$$

Dado que $P_l < P_l'$ se toma la solución $T-$.

Por tanto siempre se toma la solución negativa.

Una vez calculados todos los P_l no se tiene ninguna garantía de que la suma de estos resultados parciales dé como resultado la distancia final. Si esto no se cumple es posible que se esté colocando el punto a una distancia mayor o menor lo que aumentaría el error, para que esto no suceda se realiza un escalado de todos los P_l.

La condición que interesa que cumplan los P_l es:

$$D_{ij} = \sqrt{\sum_{l=1}^{M} (P_l \times D_{ij})^2} = \sqrt{\sum_{l=1}^{M} P_l^2} \times D_{ij} \; ; \; 1 = \sqrt{\sum_{l=1}^{M} P_l^2}$$

Por tanto el factor de escalado **X** debe ser:

$$\sqrt{\sum_{l=1}^{M} (P_l \times X)^2} = 1 \; ; \; X = \frac{1}{\sqrt{\sum_{l=1}^{M} P_l^2}}$$

2.4 ORDEN DE LOS PUNTOS

La aproximación que se ha descrito es sensible al orden en que el algoritmo recibe los puntos. Es interesante pues tratar de ordenar los puntos bajo un criterio de modo que se minimice el error cometido. Se han probado varios criterios de ordenación en concreto:

1. Sumar las distancias que un punto tiene respecto al resto de puntos y realizar una ordenación ascendente o descendente.

2. Tomar la máxima distancia que un punto tiene respecto al resto de puntos y realizar una ordenación ascendente o descendente.

Desgraciadamente ninguno de estos criterios se ha revelado como el mejor. Algunos juegos de prueba obtienen su mejor resultado con un criterio mientras que otros lo obtienen con otro criterio. De los criterios enunciados la suma de distancias en orden descendiente así como la máxima distancia en orden descendiente son los que han aportado mejores resultados. En concreto el criterio de suma de distancias en orden descendiente ha obtenido el mejor resultado con algunos juegos de pruebas mientras que con el resto no se ha alejado excesivamente de sus mejores resultados, por lo que se recomienda que se ordenen los puntos con este criterio antes de proceder a la aproximación. Como criterio suplementario en caso de que el resultado no fuera excesivamente bueno se puede utilizar el criterio de la distancia máxima en orden descendiente.

3 ESTIMACION DEL ERROR COMETIDO

Al realizar la aproximación no siempre es posible colocar los puntos a la distancia deseada por lo que se comete un error. Es interesante medir este error para saber si la aproximación dada es significativa.

Se utiliza la siguiente forma para medir el error cometido al aproximar un punto:

$$E_i = \frac{\sum_{k=1}^{N} \frac{\left| \sqrt{\sum_{j=1}^{M} (c_{ij} - c_{kj})^2} - D_{ik} \right|}{D_{ik}}}{N}$$

El error global que se comete es el error medio cometido en todos los puntos.

Las aproximaciones que se han probado han dado un error medio que oscila alrededor del 20 por cien.

4 VISUALIZACION

Este método es suficientemente general como para que el espacio al que se pasa los puntos tenga tantas dimensiones como se desee. No obstante sólo son interesantes los casos de dos y tres dimensiones. Si se pasan los puntos a un espacio de dos dimensiones podremos visualizarlos sobre papel mediante una impresora, o sobre una pantalla de ordenador. Tan sólo deben realizarse las transformaciones de escalado sobre los puntos necesarias.

Cuantas más dimensiones se tengan para colocar los puntos mayor libertad se tendrá, y por tanto menor será el error producido. Por ello es interesante representar los puntos usando tres dimensiones sobre una pantalla de ordenador.

Las tres dimensiones serán las coordenadas horizontal y vertical de los puntos de la pantalla y el color de los puntos. Se define una graduación de color, cuanto más intenso es el color más cercano se halla el punto del observador, cuanto más oscuro es el color más lejos se halla el punto.

Esta interpretación del color además se ser intuitiva corresponde a algunos estudios que han realizado algunos modelos de iluminación en los que se disminuye la claridad del color en función de la distancia de un punto al observador en [HEAR88] o [ROGE85].

5 APLICACIONES

La aplicación que se le ha dado a este método está relacionada con la clasificación automática de objetos. La clasificación automática de objetos es una técnica de la Inteligencia Artificial que trata de englobar una serie de objetos en un grupo de un modo automático. A partir de una serie de objetos y de su descripción en función de unas propiedades forma una taxonomía de clases que engloban a estos objetos.

Las propiedades que forman la descripción de un objeto podrían ser en el caso de un coche:

- Número de ruedas: 4.
- Número de caballos: 90.
- Número de puertas: 5.
- Objeto animado: NO.
- Objeto móvil: SI.

...

Si caracterizamos bajo estas propiedades a un grupo de coches, los diferentes coches variarían en el valor de algunas de estas propiedades.

Para clasificar este grupo de coches se trataría de agrupar a los coches que tengan valores y propiedades más similares.

Estos objetos forman el conjunto de puntos en el espacio **P**-dimensional que se visualizarán mediante el método anteriormente descrito permitiendo así revisar la consistencia de la clasificación. Las **P** dimensiones vienen dadas por las diferentes propiedades de los objetos mientras que las coordenadas en esas dimensiones vienen dadas por los valores de las propiedades.

6 CONCLUSIONES

1. Para representar bolas de puntos en un espacio N-dimensional se trasladan estos puntos a un espacio de dos o tres dimensiones desde el cual se puede visualizar sobre una pantalla o una impresora.

2. El paso de estos puntos de un espacio a otro implica que no se podrán conservar las distancias entre puntos y por tanto se producirá un error.

3. El método expuesto consigue unas aproximaciones que poseen un error alrededor del 20 por cien.

4. El espacio al que se pasan los puntos puede ser de tres dimensiones. Las dos primeras dimensiones corresponden a las coordenadas horizontal y vertical del punto. Como tercera dimensión se utiliza el color de los puntos. Dicho color varía en función de la proximidad o lejanía de los puntos y sirve para informar al observador de la distancia que hay entre dos puntos en la tercera dimensión.

7 BIBLIOGRAFIA

[HEAR88]: Donald Hearn, Pauline Baker. Gráficas por Computadora. Ed. Prentice Hall Hispanoamericana 1988.

[MART91]: Mario Martín Muñoz. LINNEO: Eina per a l'ajut en la construcció de bases de coneixements en dominis poc estructurats. Tesis de Licenciatura. UPC Dept. LSI. Abril 1991.

[ROGE85]: David F. Rogers. Procedural elements for computer graphics. Ed. Mc Graw Hill 1985.

APENDICE 1: CODIGO DEL PROGRAMA DE APROXIMACION

El programa de aproximación recibe el número de puntos a aproximar y su matriz de distancias y genera un fichero que guarda las coordenadas de cada punto de la aproximación. Además genera un fichero de estadísticas que lleva pro nombre ESTDCLAS.EST que contiene el error por cada punto y el error medio así como información sobre la magnitud del error. El número de dimensiones usado es tres.

```
#include <stdio.h>
#include <float.h>
#include <math.h>

#define N_DIM 3
#define RAIZ_N_DIM 1.7320508
#define MAX_PUNT 120
#define MAS 1
#define MENOS 0
#define MAX_Z 189.0
#define MIN_Z 30.0

/* Matrices para colocar las coordenadas de los puntos, las distancias entre
   estos y el sentido tomado al ajustar la distancia entre puntos */

float coord[MAX_PUNT][N_DIM];
float dist[MAX_PUNT][MAX_PUNT];
int mas_menos[MAX_PUNT], indice[MAX_PUNT];

/*****************************************************************************
*                                                                            *
*  int llenar_distancia(float dist[MAX_PUNT][MAX_PUNT],                      *
*  int indice[MAX_PUNT], char *nom_fich)                                     *
*  ------------------------------------------------------------------------- *
*                                                                            *
*  Llena la matriz de distancias con el contenido de un fichero de datos.    *
*  Devuelve en nom_fich el nombre del fichero de distancias. Indice[i] marca *
*  durante todo el programa el numero de punto inicial al que corresponde    *
*  el punto que ocupa la posicion i.                                         *
*                                                                            *
*****************************************************************************/

int llenar_distancia(float dist[MAX_PUNT][MAX_PUNT], int indice[MAX_PUNT],
                     char *nom_fich)
   {
   FILE *f; int i, j, n_puntos;

   printf("Fichero de distancias:");
   scanf("%s", nom_fich);
   f = fopen(nom_fich,"r");
   if (f == NULL)
      {
      printf("\nFichero no hallado");
      return(-1);
      }
   fscanf(f,"%d\n",&n_puntos);
   if (n_puntos > MAX_PUNT)
      return(-1);
   for(i = 0; i <n_puntos; i++)
      {
      for(j = 0; j < n_puntos; j++)
         fscanf(f,"%f ",&(dist[i][j]));
      fscanf(f,"\n");
      }
   for(i = 0; i < n_puntos; i++)
      indice[i] = i + 1;
   fclose(f);
   return(n_puntos);
```

```
   }

/*****************************************************************************
*                                                                            *
*  int guardar_puntos(float coord[MAX_PUNT][N_DIM], int n_puntos,            *
*                     int indice[MAX_PUNT])                                  *
*                                                                            *
*  ------------------------------------------------------------------------  *
*                                                                            *
*  Guarda la matriz de coordenadas en un fichero. Devuelve cero si todo va   *
*  bien y -1 sino.                                                           *
*                                                                            *
*****************************************************************************/

int guardar_puntos(float coord[MAX_PUNT][N_DIM], int n_puntos,
                   int indice[MAX_PUNT])
   {
   FILE *f; char s[30]; int i, j;

   printf("\nFichero de puntos:");
   scanf("%s", s);
   f = fopen(s,"w");
   if (f == NULL)
      {
      printf("\nEl fichero no puede abrirse");
      return(-1);
      }
   fprintf(f,"%d\n",n_puntos);
   for(i = 0; i <n_puntos; i++)
      {
      fprintf(f,"%d ",indice[i]);
      for(j = 0; j < N_DIM; j++)
         fprintf(f,"%f ",coord[i][j]);
      fprintf(f,"\n");
      }
   fclose(f);
   return(0);
   }


/*****************************************************************************
*                                                                            *
*  void calcula_error_dim(float media, int punto, int dim, float *error,     *
*                         float porc, float coord[MAX_PUNT][N_DIM],          *
*                         float dist[MAX_PUNT][MAX_PUNT])                    *
*                                                                            *
*  ------------------------------------------------------------------------  *
*                                                                            *
*  Calcula el error de los puntos que van del 0 a punto en la dimension      *
*  dim con respecto a media. porc es el porcentaje de error para esa         *
*  dimension. Devuelve el resultado en error.                                *
*                                                                            *
*****************************************************************************/

void calcula_error_dim(float media, int punto, int dim, float *error,
                       float porc, float coord[MAX_PUNT][N_DIM],
                       float dist[MAX_PUNT][MAX_PUNT])
   {
   float acu1; int i;

   *error = 0.0;
   for(i = 0; i < punto; i++)
      {

      acu1 = coord[i][dim] - media;
      acu1 = acu1 * acu1 - dist[i][punto] * dist[i][punto] * porc;
      (*error) += acu1 * acu1;
      }
   }

/*****************************************************************************
```

```
*                                                                              *
*  float calcula_error_punto(int punto, float coord[MAX_PUNT][N_DIM],          *
*                            float dist[MAX_PUNT][MAX_PUNT])                   *
*                                                                              *
*  ----------------------------------------------------------------------------*
*                                                                              *
*  Calcula el error que comete un punto que tiene las coordenadas dadas con    *
*  respecto a las distancias que se facilitan.                                 *
*                                                                              *
*******************************************************************************/

float calcula_error_punto(int punto, float coord[MAX_PUNT][N_DIM],
                          float dist[MAX_PUNT][MAX_PUNT])
   {
   float distan, error = 0, aux; int i, dim;

   for(i = 0; i < punto; i++)
      {
      distan = 0;
      for(dim = 0; dim < N_DIM; dim++)
         {
         aux = coord[i][dim] - coord[punto][dim];
         distan += aux * aux;
         }
      aux = distan - dist[i][punto] * dist[i][punto];
      error += aux * aux;
      }
   return(error);
   }


/*******************************************************************************
*                                                                              *
*  void ajusta_coordenada(float coord[MAX_PUNT][N_DIM],                        *
*                         float dist[MAX_PUNT][MAX_PUNT], int punto,           *
*                         int dim, float porc,                                 *
*                         int mas_menos[MAX_PUNT], float error[MAX_PUNT])      *
*                                                                              *
*  ----------------------------------------------------------------------------*
*                                                                              *
*  Calcula las coordenada de punto en la dimension dim que tiene proporcion    *
*  de error porc ajustandolas segun las coordenadas de los anteriores          *
*  puntos. Error indica el error acumulado en la anterior dimension que        *
*  tratara de compensarse en esta. Tambien es un parametro de salida en el     *
*  que se guarda el error acumulado en la dimension que se trata.              *
*                                                                              *
*******************************************************************************/

void ajusta_coordenada(float coord[MAX_PUNT][N_DIM],
                       float dist[MAX_PUNT][MAX_PUNT], int punto, int dim,
                       float porc, int mas_menos[MAX_PUNT],
                       float error[MAX_PUNT])
   {
   float aux, media_aux, media = 0.0, error_aux, error_act, ant_error; int i;

   for (i = 0; i < punto; i++)
      {
      mas_menos[i] = MAS;
      aux = dist[i][punto] * porc;
      aux *= aux;

      /* error pasa a tener un valor temporal para acelerar los calculos */
      if (error[i] < aux)
         {
         error[i] = sqrt(aux - error[i]);
         media += coord[i][dim] + error[i];
         }
      else
         error[i] = 0;
      }
   media /= punto;
   calcula_error_dim(media, punto, dim, &error_act, porc, coord, dist);
   do
```

```
      {
      ant_error = error_aux = error_act;
      for(i = 0; i < punto; i++)
         {
         if (mas_menos[i] == MAS)
            {
            media_aux = media - (2.0 * error[i]) / punto;
            mas_menos[i] = MENOS;
            }
         else
            {
            media_aux = media + (2.0 * error[i]) / punto;
            mas_menos[i] = MAS;
            }
         calcula_error_dim(media_aux, punto, dim, &error_aux, porc, coord,
                           dist);

         if (error_aux < error_act)
            {
            error_act = error_aux;
            media = media_aux;
            }
         else
            if (mas_menos[i] == MAS)
               mas_menos[i] = MENOS;
            else
               mas_menos[i] = MAS;
         }
      }
   while (error_act < ant_error);
   coord[punto][dim] = media;
   for (i = 0; i < punto; i++)
      {
      aux = media - coord[i][dim];
      error[i] = aux * aux - error[i] * error[i];
      }
   }


/*****************************************************************************
*                                                                            *
*  int recalcular_proporciones(float coord[MAX_PUNT][N_DIM], int punto,      *
*                        float porc[N_DIM], float dist[MAX_PUNT][MAX_PUNT]) *
*  ------------------------------------------------------------------------- *
*                                                                            *
*  Redistribuye las proporciones de error entre las dimensiones. Devuelve    *
*  cero si es posible y uno sino.                                            *
*                                                                            *
*****************************************************************************/

int recalcular_proporciones(float coord[MAX_PUNT][N_DIM], int punto,
                    float porc[N_DIM], float dist[MAX_PUNT][MAX_PUNT])
   {
   int dim, i, j, signo;
   float err_media, distan, resta_coord, sumd2c2, sumd4, sumc4;

   err_media = 0.0;

   for(dim = 0; dim < N_DIM; dim++)
      {
      sumd2c2 = sumc4 = sumd4 = 0.0;
      for(j = 0; j < punto; j++)
         for(i = j + 1; i <= punto; i++)
            {
            resta_coord = coord[i][dim] - coord[j][dim];
            resta_coord *= resta_coord;
            distan = dist[i][j] * dist[i][j];
            if (resta_coord >= distan * porc[dim])
               signo = 1;
            else
               signo = -1;
```

```
            sumd2c2 += signo * resta_coord * distan;
            sumc4 += signo * resta_coord * resta_coord;
            sumd4 += signo * distan * distan;
            }
         if (fabs(sumd4) < 1.0e-5)
            return(-1);

         resta_coord = sumd2c2 * sumd2c2 - sumd4 * (sumc4 - err_media);
         /* resta coord actua como variable auxiliar */
         if (resta_coord >= 0.0)
            if (sumd4 >= 0.0)
               porc[dim] = (sumd2c2 + sqrt(resta_coord)) / sumd4;
            else
               porc[dim] = (sumd2c2 - sqrt(resta_coord)) / sumd4;
         else
            porc[dim] = sumd2c2 /sumd4;

         if (porc[dim] < 0.0)
            porc[dim] = 0.0;
         }

      /* err_media actua como variable auxiliar */
      err_media = 0.0;
      for(i = 0; i < N_DIM; i++)
           err_media += porc[i] * porc[i];
      err_media = 1.0 / sqrt(err_media);
      for(i = 0; i < N_DIM; i++)
         porc[i] *= err_media;
      return(0);
      }


/*****************************************************************************
*                                                                            *
*  void coordenadas(float coord[MAX_PUNT][N_DIM], int punto, int ult_punto) *
*  _________________________________________________________________________ *
*                                                                            *
*  Determina las coordenadas de punto en funcion de los puntos 0 a           *
*  ult_punto, si la raiz de la ultima coordenada es negativa la ultima       *
*  coordenada se coloca a cero.                                              *
*                                                                            *
*****************************************************************************/

void coordenadas(float coord[MAX_PUNT][N_DIM], int punto, int ult_punto)
   {
   int i, j; float aux;

   for(j = 1; j <= ult_punto; j++)
         {
         aux = 0.0;
         for(i = 1; i <= j - 1; i++)
            aux += coord[j][i - 1] * (coord[j][i - 1] -
                                      2.0 * coord[punto][i - 1]);
         aux += coord[j][j - 1] * coord[j][j - 1];
         aux += dist[0][punto] * dist[0][punto] -
                dist[j][punto] * dist[j][punto];
         coord[punto][j - 1] = aux / (2.0 * coord[j][j - 1]);
         }
      aux = 0.0;
      for(i = 1; i <= ult_punto; i++)
         aux += coord[punto][i - 1] * coord[punto][i - 1];
      if (aux >= dist[0][punto] * dist[0][punto])
         coord[punto][ult_punto] = 0.0;
      else
         coord[punto][ult_punto] = sqrt(dist[0][punto] * dist[0][punto]
                                        - aux);
      for(j = ult_punto + 1; j < N_DIM; j++)
         coord[punto][j] = 0.0;
   }
```

```
/*****************************************************************************
*                                                                            *
*  void intercambiar_columnas(float coord[MAX_PUNT][N_DIM],                  *
*                             float dist[MAX_PUNT][MAX_PUNT], int col1,      *
*                             int col2, int n_puntos, int indice[MAX_PUNT]) *
*  ------------------------------------------------------------------------  *
*                                                                            *
*  Intecambia las columnas col1 y col2 de la matriz coord, las               *
*  correspondientes filas y columnas de la matriz dist y el vector indice.   *
*                                                                            *
*****************************************************************************/

void intercambiar_columnas(float coord[MAX_PUNT][N_DIM],
                           float dist[MAX_PUNT][MAX_PUNT], int col1,
                           int col2, int n_puntos, int indice[MAX_PUNT])

   {
   int i; float aux;

   for(i = 0; i < N_DIM; i++)
      {
      aux = coord[col1][i];
      coord[col1][i] = coord[col2][i];
      coord[col2][i] = aux;
      }
   for(i = 0; i < n_puntos; i++)
      {
      aux = dist[col1][i];
      dist[col1][i] = dist[col2][i];
      dist[col2][i] = aux;
      }
   for(i = 0; i < n_puntos; i++)
      {
      aux = dist[i][col1];
      dist[i][col1] = dist[i][col2];
      dist[i][col2] = aux;
      }
   aux = indice[col1];
   indice[col1] = indice[col2];
   indice[col2] = aux;
   }


/*****************************************************************************
*                                                                            *
*  void colocar_primeros(float coord[MAX_PUNT][N_DIM],                       *
*                        float dist[MAX_PUNT][MAX_PUNT], int n_puntos,       *
*                        int *cantidad, int indice[MAX_PUNT])                *
*  ------------------------------------------------------------------------  *
*                                                                            *
*  Coloca los primeros N_DIM puntos. n_puntos indica el numero de puntos     *
*  existentes. Devuelve en cantidad el numero de puntos que ha conseguido    *
*  colocar. Guarda los puntos ya calculados a partir del punto N_DIM + 1     *
*  puede darse el caso de que se agoten los puntos a calcular, en ese caso   *
*  reemplaza los puntos que son menores de N_DIM y que debieran haberse      *
*  calculado directamente, en este caso se acaban por calcular todos los     *
*  puntos.                                                                   *
*                                                                            *
*****************************************************************************/

void colocar_primeros(float coord[MAX_PUNT][N_DIM],
                      float dist[MAX_PUNT][MAX_PUNT], int n_puntos,
                      int *cantidad, int indice[MAX_PUNT])
   {
   int i, k, final, a_colocar = N_DIM + 1, incr = 1;

   if (a_colocar > n_puntos - 1)
      a_colocar = n_puntos - 1;  /* Por si hay pocos puntos */

   for(i = 0; i < N_DIM; i ++)
      coord[0][i] = 0.0;     /* primer punto en el 0,0 ... 0 */
```

```
   final = 0;
   k = 1;
   while (! final)
      {
      coordenadas(coord, k, k - 1);
      if (fabs(coord[k][k-1] * coord[k][k-1]) < 1.0e-5)
         /* dado que coord es el resultado de una raiz, esta raiz podria
            enmascarar una posible imprecision, por lo que al comprobar
            si hay imprecision se eleva al cuadrado */
         {
         intercambiar_columnas(coord, dist, a_colocar, k, n_puntos, indice);
         if (a_colocar == n_puntos - 1)
            {   /* Se han acabado los puntos a calcular */
            incr = -1;
            a_colocar = N_DIM;
            if (a_colocar > n_puntos - 2)
               a_colocar = n_puntos - 2;
            }
         else
            a_colocar += incr;
         }
      else
         k ++;
      final = (k >= N_DIM + 1) || (a_colocar < k);
      }
   if (a_colocar < k)
      *cantidad = n_puntos;
   else
      *cantidad = a_colocar;
   }


/*****************************************************************************
*                                                                            *
*  void ubicacion(float dist[MAX_PUNT][MAX_PUNT],                            *
*                 float coord[MAX_PUNT][N_DIM], int mas_menos[MAX_PUNT],     *
*                 int n_puntos, int indice[MAX_PUNT])                        *
*                                                                            *
*  ------------------------------------------------------------------------  *
*                                                                            *
*  Coloca los puntos en la matriz coord tomando las distancias de dist.      *
*  n_puntos indica el numero de puntos existentes.                           *
*                                                                            *
*****************************************************************************/

void ubicacion(float dist[MAX_PUNT][MAX_PUNT],
               float coord[MAX_PUNT][N_DIM], int mas_menos[MAX_PUNT],
               int n_puntos, int indice[MAX_PUNT])
   {

   int i, j, n, mal; float error_ant, error_act;
   float porc[N_DIM], error[MAX_PUNT], coord2[N_DIM],
   ini_porc = 1.0 / RAIZ_N_DIM;

   printf("\n");
   colocar_primeros(coord, dist, n_puntos, &n, indice);

   for(i = n; i < n_puntos; i++)
      {
      for(j = 0; j < N_DIM; j++)
         porc[j] = ini_porc;

      for(j = 0; j < i; j++)
         error[j] = 0.0;
      for(j = 0; j < N_DIM; j++)
         ajusta_coordenada(coord, dist, i, j, porc[j], mas_menos, error);
      error_act = calcula_error_punto(i, coord, dist);
      do
         {
         error_ant = error_act;
         for(j = 0; j < N_DIM; j++)
            coord2[j] = coord[i][j];
```

```
        mal = recalcular_proporciones(coord, i, porc, dist);

        for(j = 0; j < i; j++)
           error[j] = 0.0;
        for(j = 0; j < N_DIM; j++)
           ajusta_coordenada(coord, dist, i, j, porc[j], mas_menos, error);

        error_act = calcula_error_punto(i, coord, dist);
        }
     while ((error_act < error_ant) && (mal == 0));
     for(j = 0; j < N_DIM; j++)
        coord[i][j] = coord2[j];
     printf(".");
     }
  }


/*****************************************************************************
*                                                                            *
*  void estadisticas(float coord[MAX_PUNT][N_DIM],                           *
*                    float dist[MAX_PUNT][MAX_PUNT], int indice[MAX_PUNT],   *
*                    int n_puntos, char *nom_fich, int orden)                *
*                                                                            *
*  ------------------------------------------------------------------------  *
*                                                                            *
*  Calcula las estadisticas de la clasificacion (error) y las gurada en el   *
*  fichero ESTDCLAS.EST.                                                     *
*                                                                            *
*****************************************************************************/

void estadisticas(float coord[MAX_PUNT][N_DIM],
                  float dist[MAX_PUNT][MAX_PUNT], int indice[MAX_PUNT],
                  int n_puntos, char *nom_fich, int orden)

  {
  int i, j, k; float acu, aux, suma_e, suma_d, suma_tot_error, suma_error;
  FILE *f;

  f = fopen("ESTDCLAS.EST","w");

  fprintf(f,"Estadisticas del Fichero %s\n", nom_fich);
  if (orden == 1)
     fprintf(f,"Ordenacion por suma de distancias\n\n");
  else
     fprintf(f,"Ordenacion por distancia maxima\n\n");
  suma_tot_error = 0.0;
  for(i = 0; i < n_puntos; i++)
     {
     suma_d = suma_e = suma_error = 0.0;
     for(j = 0; j < n_puntos; j++)
        {
        acu = 0.0;
        for(k = 0; k < N_DIM; k++)
           {
           aux = coord[i][k] - coord[j][k];
           acu += aux * aux;
           }
        if (dist[i][j] > 0.0)
           {
           suma_d += dist[i][j];
           aux = fabs(sqrt(acu) - dist[i][j]);
           suma_error += aux;
           suma_e += aux / dist[i][j];
           }
        }
     suma_e = suma_e / (float) n_puntos;
     fprintf(f,"Punto: %d, error: %f Error abs.: %f Suma d. %f\n", indice[i],
             suma_e, suma_error, suma_d);
     suma_tot_error += suma_e;
     }
  fprintf(f,"\nError global : %f\n",
          fabs(suma_tot_error / (float) n_puntos));
```

```
   fclose(f);
   }


/*****************************************************************************
*                                                                            *
*  void ordenar(float dist[MAX_PUNT][MAX_PUNT], int indice[MAX_PUNT],        *
*               int orden, int n_puntos, float coord[MAX_PUNT][N_DIM])       *
*  ------------------------------------------------------------------------  *
*                                                                            *
*  Reordena los puntos en funcion de orden. Si orden es 1 los ordena por     *
*  suma de distancias ascendente, si es 2 los ordena por distancia maxima    *
*  ascendente.                                                               *
*                                                                            *
*****************************************************************************/

void ordenar(float dist[MAX_PUNT][MAX_PUNT], int indice[MAX_PUNT], int orden,
             int n_puntos, float coord[MAX_PUNT][N_DIM])

   {
   int i, j; float aux[MAX_PUNT], aaux;

   for(i = 0; i < n_puntos; i++)
      {
      aux[i] = dist[i][0];
      for (j = 1; j < n_puntos; j++)
         if (orden == 1)
            aux[i] += dist[i][j];
         else
            if (dist[i][j] > aux[i])
               aux[i] = dist[i][j];
      }

   for(i = 0; i < n_puntos - 1; i++)
      for (j = i + 1; j < n_puntos; j++)
         if (aux[i] < aux[j])
            {
            intercambiar_columnas(coord, dist, i, j, n_puntos, indice);
            aaux = aux[i];
            aux[i] = aux[j];
            aux[j] = aaux;
            }
   }


void main(int argc, char *argv[])
   {
   int n_puntos, orden; char nom_fich[50];

   if (argc == 2)
      sscanf(argv[1],"%d",&orden);
   else
      orden = 1;
   if ((orden < 0) || (orden > 2))
      {
      printf("\nOrden incorrecto");
      return;
      }

   n_puntos = llenar_distancia(dist, indice, nom_fich);
   if (n_puntos == -1)
      return;

   ordenar(dist, indice, orden, n_puntos, coord);
   ubicacion(dist, coord, mas_menos, n_puntos, indice);
   guardar_puntos(coord, n_puntos, indice);
   estadisticas(coord, dist, indice, n_puntos, nom_fich, orden);
   }
```

APENDICE 2: RESULTADOS DE LA APROXIMACION

Fichero inicial de distancias llamado comp.dis:

```
8
0.0 1.506266 1.991487 2.217587 1.846897 1.390729 2.991526 1.115983
1.506266 0.0 0.970935 0.911321 0.993012 1.980913 1.48526 0.64305
1.991487 0.970935 0.0 0.859615 0.844591 2.084373 1.200038 0.875505
2.217587 0.911321 0.859615 0.0 0.540182 2.310473 1.773939 1.135503
1.846897 0.993012 0.844591 0.540182 0.0 2.090727 2.044629 0.730915
1.390729 1.980913 2.084373 2.310473 2.090727 0.0 2.617745 1.605095
2.991526 1.48526 1.200038 1.773939 2.044629 2.617745 0.0 2.075543
1.115983 0.64305 0.875505 1.135503 0.730915 1.605095 2.075543 0.0
```

Fichero de resultados generado por el programa:

```
8
7 0.000000 0.000000 0.000000
6 2.617745 0.000000 0.000000
1 2.648785 1.390383 0.000000
4 0.890301 0.885354 1.253145
5 1.286866 1.312189 0.769495
3 0.777151 0.779687 0.502431
2 1.022369 0.588056 0.500475
8 1.711003 0.644898 0.591290
```

El primer número indica el número de punto del que se trata.

Fichero de estadísticas generado por el programa:

```
Estadisticas del Fichero comp.dis
Ordenacion por suma de distancias

Punto: 7, error: 0.030670 Error abs.: 0.420049 Suma d. 14.188681
Punto: 6, error: 0.045581 Error abs.: 0.646350 Suma d. 14.080055
Punto: 1, error: 0.077296 Error abs.: 0.916087 Suma d. 13.060473
Punto: 4, error: 0.082247 Error abs.: 0.455595 Suma d. 9.748620
Punto: 5, error: 0.121357 Error abs.: 0.936046 Suma d. 9.090954
Punto: 3, error: 0.122720 Error abs.: 0.959268 Suma d. 8.826544
Punto: 2, error: 0.191606 Error abs.: 1.769032 Suma d. 8.490757
Punto: 8, error: 0.101123 Error abs.: 0.978620 Suma d. 8.181594

Error global : 0.096575
```

La primera columna indica el error, la segunda indica el valor del error sin promediar (es decir sin dividir la diferencia de distancias por la distancia inicial). La tercera columna indica la suma de todas las distancias de este punto con respecto al resto de puntos y se incluye como referencia de la segunda columna.