Sistemas Informáticos
Curso 06-07

Herramienta abierta de regresión borrosa

Javier Rivas Rodríguez
Manuel Marcos Vega

Dirigido por:
Prof. F.Javier Crespo Yañez
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

1. Resumen

El uso de las técnicas de regresión sobre las observaciones experimentales ha permitido el estudio de numerosos fenómenos en diversos campos de la ciencia, y muy especialmente en las ciencias sociales. Dichas técnicas requieren de un número suficiente de observaciones “precisas”, exactas y fiables. Sin embargo, no siempre es posible obtener el conjunto de observaciones necesario, o éstas contienen algún tipo de imperfección en los datos, debido a la imprecisión o vaguedad de los mismos.
En cualquier caso, con suficientes datos o no, con imperfecciones o no, los modelos obtenidos deberían proveer de capacidades predictivas y descriptivas [JCr02]. Las actuales herramientas, o las más fácilmente accesibles, tienen limitado el uso de modelos y difícilmente usan las técnicas de la teoría de conjuntos borrosos.

Se propone en este trabajo una herramienta abierta de regresión que admita el uso de cualquier modelo de curva independientemente de su naturaleza. Además, esta herramienta permitirá el uso de diferentes formas de borrosidad y por su diseño permitiría cualquier modelo propuesto por el usuario si éste prevee que éstos tienen características que sean suficientemente predictivas y descriptivas.
Esta primera aproximación de una herramienta abierta de regresión se realiza un estudio sobre diferentes modelos paramétricos simbólicos, usados comúnmente en la práctica en disciplinas tan heterogéneas como pueden ser la Ingeniería del Software, la Economía o en cualquier campo en donde puedan aparecer imprecisiones en la información.

Abstract

The use of regression techniques in experimental observations has led to the study of numerous phenomena in various fields of science, especially in social science. These techniques require a sufficient number of “precise”, exact and reliable observations. However, it is not always possible to obtain all the necessary group of observations or these have some failings, as a result of inexact or vague data.
Nevertheless, having more or less data, with or without failings, the obtained paradigms should provide predictive and descriptive capacities. The current tools or those more accessible have limited paradigm application and hardly use the techniques relating the fuzzy sets theory.
In this first approach to an open regression tool, a study has been carried out of the different parametric, symbolic paradigms, commonly used in the practice of such diverse disciplines as Software Engineering, Economy or any other field where information imprecision can appear.

2. Palabras Clave

Regresión borrosa, Ajuste de curvas, Lógica borrosa, modelos matemáticos simbólicos, imprecisión, herramienta abierta, motor regresión, herramienta abierta, agregador, distancia, parámetros.

3. Autorización

Autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

4. Introducción

El hombre tiene la capacidad de percibir el mundo que le rodea y expresarlo mediante el lenguaje natural, dicho lenguaje es una forma de comunicación imprecisa y ambigua que se apoya en el conocimiento compartido por aquellos con los que se comunica [Jlaw01].

La mayoría de los fenómenos que encontramos cada día son imprecisos, es decir, tienen implícito un cierto grado de vaguedad en la descripción de su naturaleza. Esta imprecisión puede estar asociada con su forma, posición, momento, color, textura, o incluso en la semántica que describe lo que son [BCF05]. En muchos casos el mismo concepto puede tener diferentes grados de imprecisión en diferentes contextos o tiempo. ¿Qué es una persona alta? ¿A partir de qué edad una persona deja de ser joven?

Según la bibliografía consultada, entre los modelos numéricos propuestos para enfrentarse a este tipo de problemas, uno de los más empleados es la teoría de conjuntos borrosos. La borrosidad es la propiedad asociada al uso de predicados inciertos, como “María es guapa”. Se fundamenta en la pertenencia de un elemento a un conjunto, un grado que no tiene porqué ser necesariamente ‘0’ o ‘1’ como en el caso de la teoría de conjuntos clásica [BCF05].Los valores intermedios son admitidos para hacer frente a estos otros casos. De tal forma, la borrosidad nos aclara acerca de la imprecisión derivada de la pertenencia parcial de un elemento dado y bien definido a un conjunto cuyos extremos no están definidos de una forma clara. Este tipo de imprecisión asociada continuamente a los fenómenos es común en todos los campos de estudio: sociología, física, biología, finanzas, ingeniería, oceanografía, psicología, etc. [Calv03].

Hay que tener en cuenta que la idea en sí de que las cosas no son blancas o negras, sino que existen infinitos matices de grises viene ya desde la época de los primeros grandes filósofos como Platón. Posteriormente a ellos, otros grandes pensadores como David Hume o Kant apoyaban esta idea manteniendo que el razonamiento venía dado por las observaciones de las que somos testigos a lo largo de nuestra vida y la detección de algunos principios contradictorios en la lógica clásica.[WIK]

El inicio de la lógica difusa se sitúa en 1965 con Lotfi Zadeh, profesor de ciencia de computadoras en la Universidad de California de Berkeley, guiado por el principio de que las matemáticas pueden ser usadas para encadenar el lenguaje con la inteligencia humana [ACIS3].

La lógica "fuzzy" o difusa es básicamente una lógica multievaluada que permite valores intermedios para poder definir evaluaciones convencionales como si / no, verdadero/ falso, negro/ blanco, etc [BCF05]. La lógica difusa es un lenguaje que permite trasladar sentencias sofisticadas del lenguaje natural a un formalismo matemático [ACIS3].

En general, la lógica difusa es aplicada en cualquier campo donde haya modelos no lineales donde las aproximaciones sean difíciles, en sistemas controlados por expertos humanos, en sistemas donde se tienen entradas y salidas que son continuas y complejas o en sistemas que utilizan principalmente observaciones humanas como entradas o reglas básicas [ACIS3]. De esta forma se ha realizado un intento de aplicar una forma más humana de pensar en la programación de computadoras [BCF05].

La teoría de los conjuntos difusos da flexibilidad a la modelización utilizando variables lingüísticas. En el apéndice B está la teoría necesaria para entender este tipo de conjuntos.

En la actualidad es un campo de investigación con un gran número de científicos trabajando en ello, atraídos quizás por sus aplicaciones prácticas para procesos no lineales en las que no existe un modelo de solución sencillo. [Gal05].

El Análisis de Regresión es una técnica estadística que tiene como objetivo establecer modelos matemáticos para representar formalmente las relaciones de dependencia existente entre un conjunto de variables estadísticas. Tanto en el caso de dos variables (regresión simple) como en el de más de dos variables (regresión múltiple), el análisis de regresión lineal puede utilizarse para explorar y cuantificar la relación entre una variable llamada dependiente o criterio (Y) y una o más variables llamadas independientes o predictoras (X1, X2, ..., Xk) [RLM00].

Según la literatura consultada, el análisis de regresión se utiliza para predecir un amplio rango de fenómenos, desde medidas económicas hasta diferentes aspectos del comportamiento humano. Uno de los principales usos puede encontrarse aplicado a estudios referentes a poblaciones, ya sean económicos, electorales, de producción, etc...

Los modelos paramétricos matemáticos son parte de las regresiones. Los más conocidos que se han investigado trabajan con números crisp (nítidos), obviando la ambigüedad en la precisión de los datos. En el caso de que el objetivo sea conseguir resultados más representativos, es necesario usar modelos paramétricos matemáticos simbólicos de carácter predictivo y descriptivo [JCr02]. En los modelos donde los datos son insuficientes o imperfectos, originados por la imprecisión, se ha demostrado útil el uso de un análisis borroso [ChA01], [BCF05].

El análisis de regresión borroso ha sido estudiado y aplicado en diferentes áreas tal como el modelado de datos económicos o financieros [CDS86], y se han obtenido resultados comparables y que aportan ajustes, en muchos casos superiores, a los obtenidos con el análisis de regresión clásico. Sin embargo, resulta extraño la dificultad de encontrar herramientas que permitan como entrada diferentes formas de borrosidad, con el fin de obtener una mayor descripción de la información que se pretende representar.

Las herramientas desarrolladas actualmente para el trato de este tipo de regresiones no cubren la funcionalidad necesaria que un ámbito de estas características requiere.

En el diseño de una herramienta computacional es muy importante tratar de diseñarla con expectativas futuras. Siempre aparecen nuevos retos que solucionar e incorporar, mejoras a las funcionalidades existentes o aspectos que antes no se habían tenido en cuenta y que empiezan a tomar relevancia.

Con un diseño abierto la aplicación tendrá una organización tal que en caso de querer introducir nuevas funcionalidades se conseguirá de forma rápida y nada engorrosa y el cambio en ésta será mínimo.

Las consecuencias de que una herramienta sea abierta radica en los beneficios que a largo plazo pueden suponer en aspectos como el ahorro de tiempo y coste. Es cierto que el diseño requiere más tiempo, pero el que se ganará en el futuro y los errores que se pueden reducir hacen que una herramienta calificada como abierta sea más interesante.

El principal objetivo de este proyecto es la realización del diseño de la aplicación utilizando las tecnologías más avanzadas. Tendrá un motor capaz de realizar regresiones usando lógica difusa. Entendemos el motor como una clase que decidirá que objetos crear para invocar los métodos necesarios y realizar la regresión dependiendo de las opciones introducidas por el usuario.

Otro de los objetivos que se pretende alcanzar con la implementación de este sistema es la realización de un diseño tal que las líneas de investigación futuras no tengan que tener en cuenta la implementación del sistema y que la adicción de nuevas funcionalidades se haga con total facilidad.

A la hora de diseñar la aplicación tomaremos una metodología orientada a objetos.

Realizaremos un estudio de como interactúan las clases del sistema entre sí utilizando una herramienta de diseño, Rational Rose.

El diseño de la interfaz gráfica debe hacer que ésta sea lo más sencilla posible y que la interacción del usuario con el programa sea mínima.

Verificar que este diseño es idóneo y que el motor puede funcionar lo determinará la implementación del código necesario para realizar una regresión utilizando una curva sencilla y todos los métodos necesarios para poder operar con números borrosos.

Para la realización de las pruebas que demuestren el paso anterior necesitamos crear una serie de puntos para hacer la regresión. Podemos crearlos en una clase adecuándose a la representación que elijamos, pero además ofreceremos una posible forma de introducir puntos borrosos a través de una interfaz gráfica y demostraremos que es válida.

Por otra parte la supervisación del proyecto a distancia, supondrá un handicap que solucionaremos con comunicación vía email.

5. Introducción a la teoría de lógica borrosa

5.1. Herramientas Abiertas.

Una herramienta abierta debe tener un diseño que permita que cualquier mejora futura sea fácilmente implementable e introducida en la aplicación.

En el diseño de una herramienta informática es muy importante tratar de diseñarla con expectativas futuras, es decir, teniendo en cuenta que todo evoluciona y puede mejorar.

Salvo situaciones físicas y problemas complejos sin solución pocas cosas son imposibles. Siempre aparecen nuevos retos que solucionar e incorporar, mejoras a las funcionalidades existentes o aspectos que antes no se habían tenido en cuenta y que empiezan a tomar relevancia.

Por tanto, en caso de querer introducir nuevas funcionalidades a la herramienta, el diseño inicial de la aplicación debe ser “abierto” para que la introducción sea más fácil y segura.

Las consecuencias de que una herramienta sea abierta radica en los beneficios que a largo plazo pueden suponer.

Es cierto que el diseño se hace más pesado, sabemos que no es “divertido” la apariencia genérica de las clases que forman la aplicación pero el tiempo que se ganará y los errores futuros que se pueden reducir hacen que una herramienta calificada como abierta aumente el interés sobre los posteriores desarrolladores.

5.2. Una herramienta de regresión borrosa

En el objetivo de conseguir una aplicación que realice regresiones usando lógica borrosa, hemos tenido en cuenta que los métodos y clases utilizados son susceptibles de posteriores ampliaciones y mejoras. Por ello, se ha realizado un diseño de cara a: nuevas curvas, métodos de cálculo de distancias, avance de los parámetros, etc.

Usando técnicas de ingeniería del software hemos conseguido una organización del código adecuada a esta demanda.

6. Proceso de Desarrollo

6.1. Introducción a la Ingeniería del Software

Ingeniería del software es: El desarrollo y el uso de principios correctos de ingeniería para obtener de manera económica software fiable, mantenible y eficaz.

Tras los conocimientos adquiridos con este tipo de ingeniería podemos tomar dos puntos de vista diferentes en referencia a la importancia de su estudio y utilidad.

Es cierto que una gran parte puede resultar demasiado formal a la hora de realizar nuestro proyecto, incluso innecesaria. Sin embargo tres aspectos destacan claramente:

- La importancia del uso de un modelo.
- La organización del proyecto en capas.
- El uso de patrones.
-

La madurez "informática" de un ingeniero está en la predisposición y manera en que afronta un problema. La forma en la que organiza y busca una solución.

Cuando se carece de experiencia en el diseño e implementación de proyectos el principal error es empezar codificando las posibles soluciones para resolver el problema sin pararse a pensar en un buen diseño que facilite posteriormente el desarrollo, en que nos hace falta y en que no, en los riesgos y el coste.

Utilizando un modelo conseguimos que la implementación del código sea el paso menos importante. Se debería concienciar a los primeros cursos que con un buen diseño, la implementación es prácticamente automática. Prueba de ello es el uso de herramientas como "Rational Rose" la cual explicaremos más adelante en relación a nuestro proyecto.

El modelo que más se aproxima al que hemos seguido se denomina "Modelo secuencial".

Cuatro partes forman este modelo.
1. Análisis.
2. Diseño.
3. Implementación.
4. Pruebas.

6.2. Análisis

En este bloque se han llevado a cabo las siguientes tareas:

- Funcionalidad del sistema, versiones anteriores:

Comienzo de una serie de reuniones con el coordinador en las que se expone el problema a tratar. Se debe sacar en claro que debe hacer el sistema y las posibles vías para implementarlo.

Puesto que nuestro sistema es una continuación en la investigación de la lógica borrosa se proporciona toda la documentación hasta la fecha para poder situarnos en el nivel de conocimientos suficiente.

- Comprensión del problema:

Ayudándonos de la documentación se debe entender el dominio del problema.

- Brainstorming (tormenta de ideas)

En esta parte se realiza un bombardeo de posibles soluciones y formas que puede tomar el sistema de cara a empezar a diseñar el sistema.

6.3. Diseño

6.3.1. Idea general

Las curvas sobre las que se va a hacer la regresión tienen parámetros, y son éstos los que deberán variar para ajustarse lo más posible a los puntos introducidos por el usuario de la herramienta.

Los parámetros de la curva pueden determinarse de dos maneras, en función de la decisión del usuario. Éste puede dar los parámetros iniciales de la curva que ha elegido, o por el contrario, puede dejar que la herramienta genere unos por defecto.

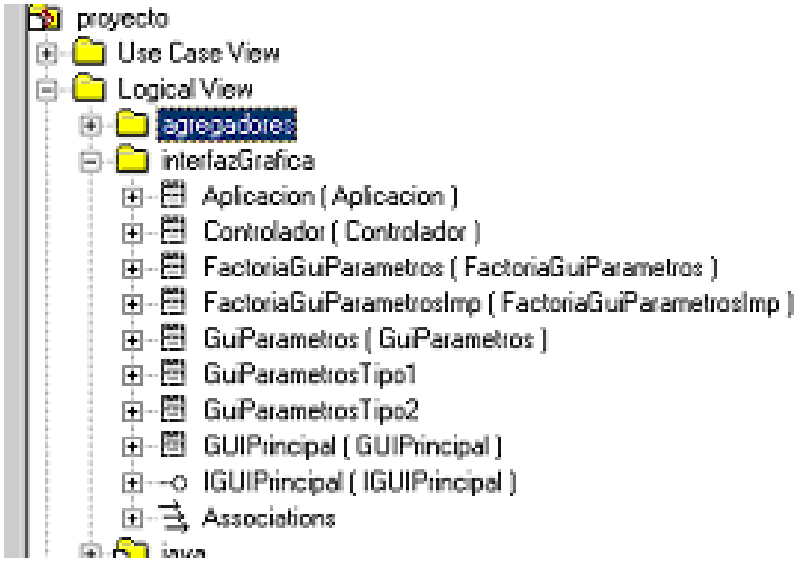
Una vez que estos parámetros son elegidos, la idea consiste en ir variándolos (en función de determinados algoritmos de variación de parámetros) hasta conseguir unos que se ajusten lo más posible a los puntos que ha introducido el usuario.

La decisión de qué parámetros son los más adecuados viene dado por la distancia de los puntos a la curva. Aquellos parámetros que hagan mínima la distancia de los puntos a la curva serán los más apropiados y determinarán unívocamente la curva que más se ajusta a los puntos introducidos por el usuario.

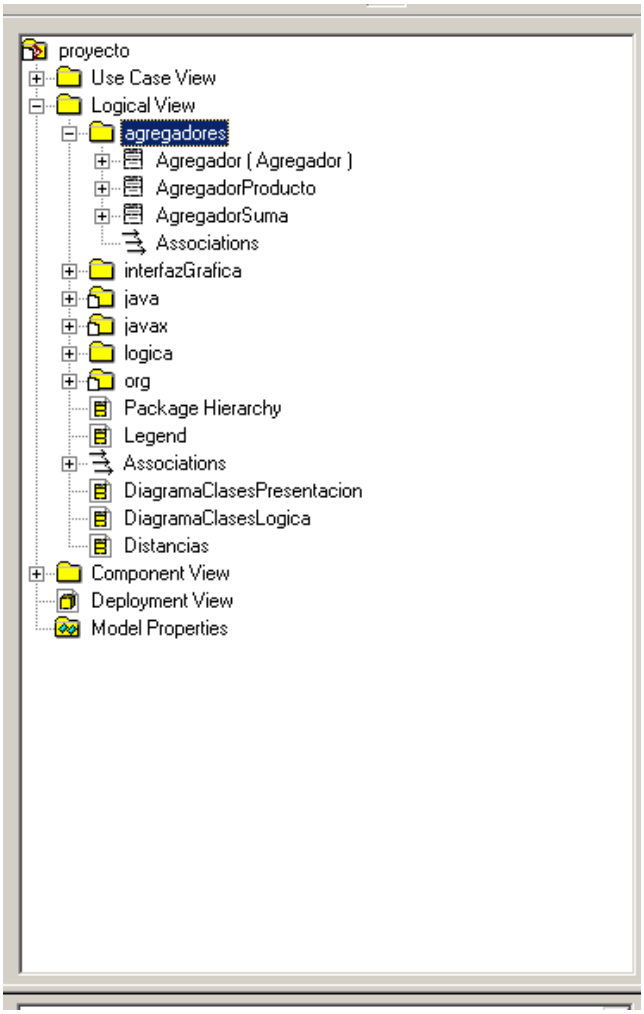
6.3.2. Descripción del diseño

Con el uso del Rational Software llegamos finalmente a la siguiente organización del proyecto.

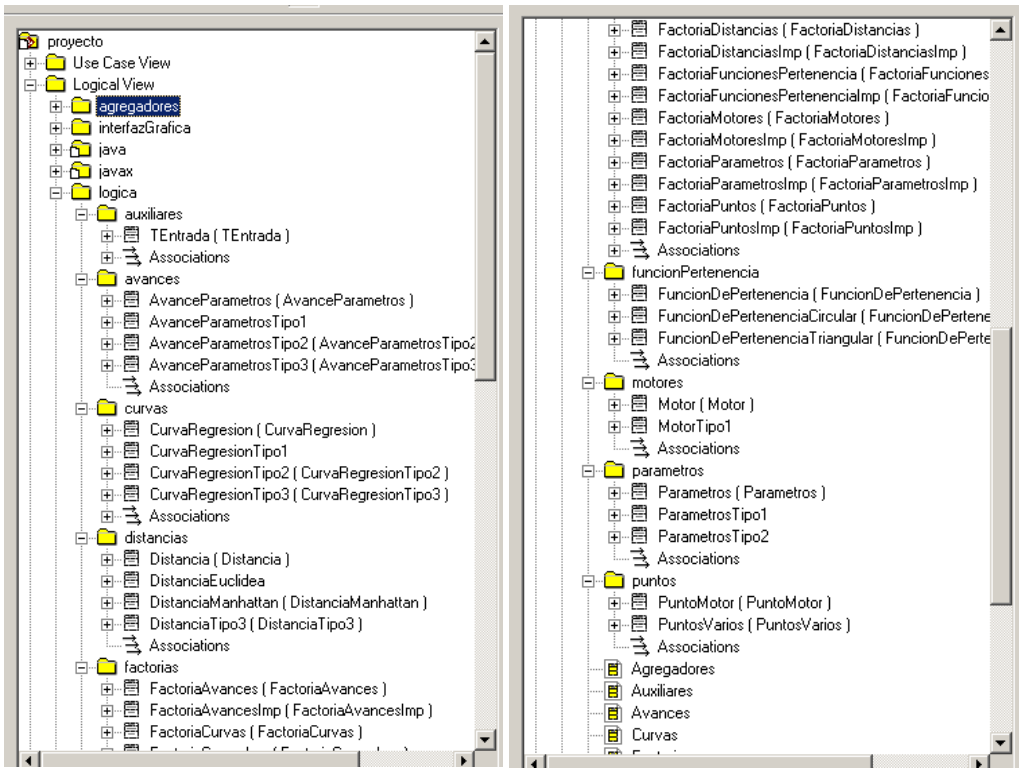
En el paquete de interfaz gráfica están las clases relacionadas con toda la capa de presentación que interactúan con el usuario antes de que el motor de regresión se ponga en funcionamiento.



En el paquete agregadores se encuentran todas las clases relacionadas con las operaciones entre números borrosos. Hasta el momento hemos necesitado implementar solo el agregador que realice la suma y el agregador que realiza el producto:



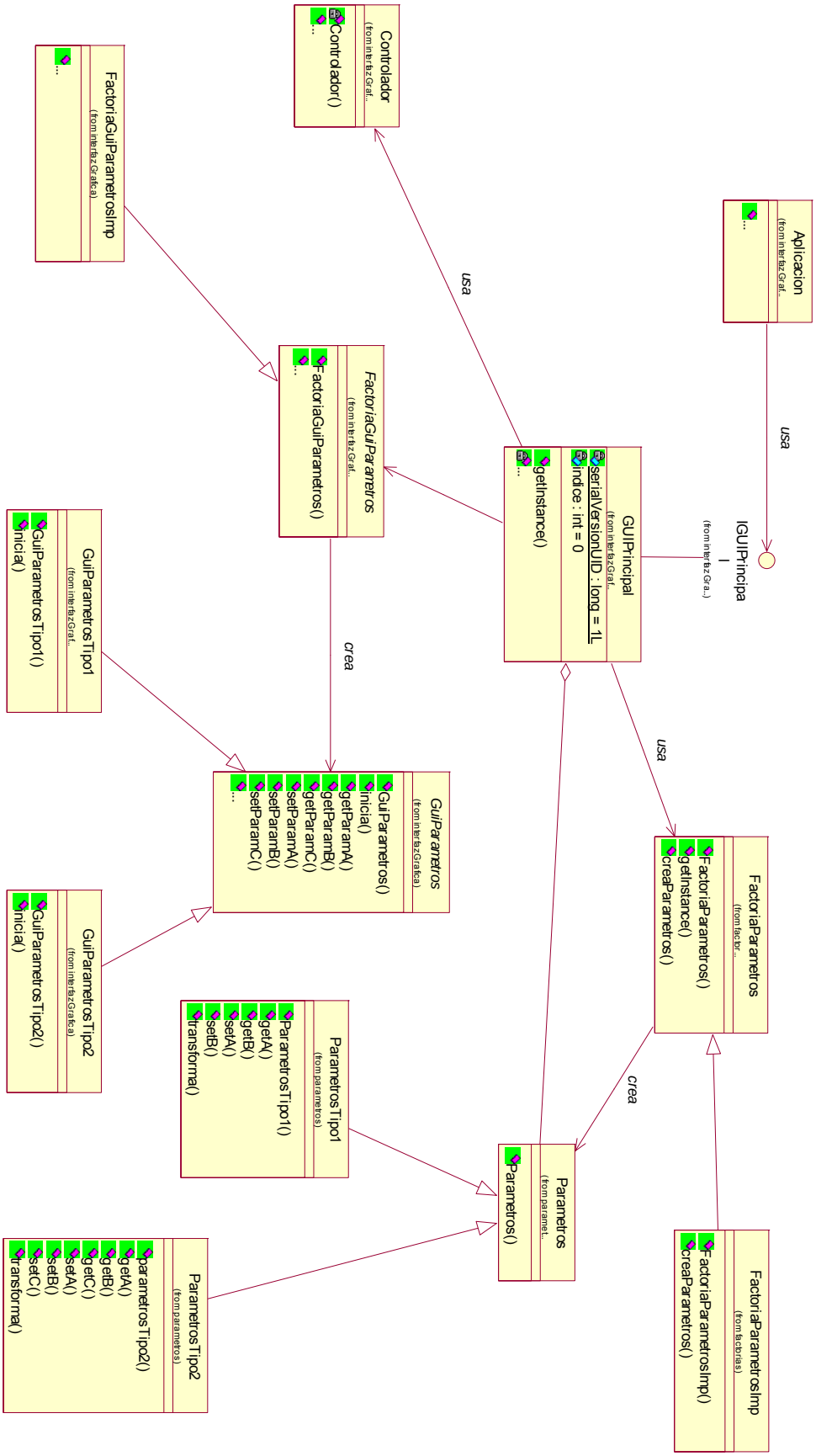
Finalmente el paquete lógica tiene las clases necesarias para que el motor de regresión funcione.



6.3.3. Capa de presentación

Por una parte tenemos todas las clases referentes a la interfaz gráfica de usuario y por otra la lógica necesaria para realizar los cálculos elegidos.

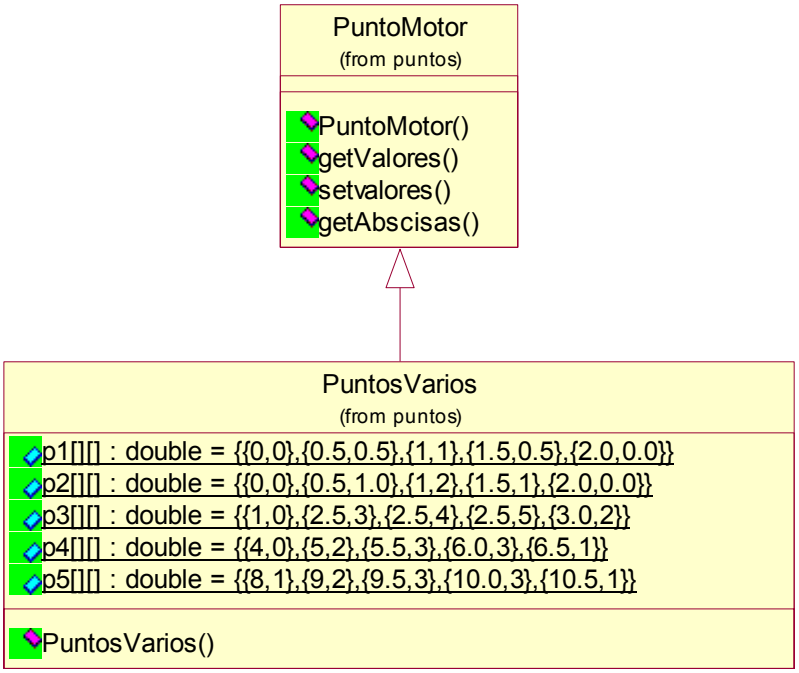
El diagrama de clases para la capa de presentación es el siguiente:


Aplicacion
usa
IGUIPrincipa
GUIPrincipal
serialVersionUID : long = 1L
indice : int = 0
getInstance()
usa
Controlador
Controlador()
usa
FactoriaGuiParametrosImp
FactoriaGuiParametros
FactoriaGuiParametros()
crea
GuiParametrosTipo1
GuiParametrosTipo1()
inicia()
GuiParametros
GuiParametros()
inicia()
getParamA()
getParamB()
getParamC()
setParamA()
setParamB()
setParamC()
GuiParametrosTipo2
GuiParametrosTipo2()
inicia()
usa
FactoriaParametros
FactoriaParametros()
getInstance()
creaParametros()
crea
ParametrosTipo1
ParametrosTipo1()
getA()
getB()
setA()
setB()
transforma()
Parametros
Parametros()
FactoriaParametrosImp
FactoriaParametrosImp()
creaParametros()
ParametrosTipo2
parametrosTipo2()
getA()
getB()
getC()
setA()
setB()
setC()
transforma()

Para especificar el diseño de la parte relacionada con toda la lógica del proyecto, en primer lugar mostramos los diagramas de clases de cada paquete que lo forma así como una explicación de lo que hace cada clase en sí.

6.3.3.1. Puntos

Diagrama de clases.

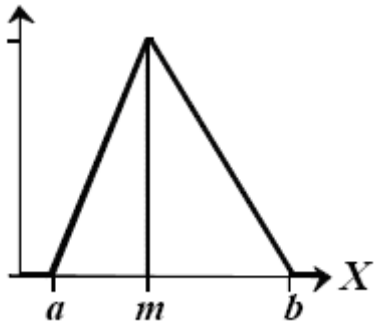


Punto Motor

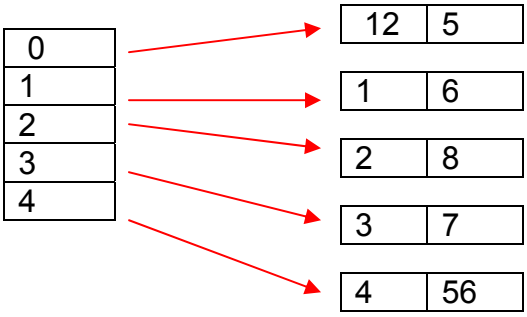
Esta clase representa la estructura de datos que maneja el motor para operar con puntos borrosos.
Cómo representar los números borrosos para poder operar con ellos de una manera sencilla ha representado un problema bastante importante de cara a la implementación del sistema.

Partiendo de que un punto es borroso si alguna de sus componentes es borrosa, y viene definido por un centro y una función de pertenencia (ver Apendice B), esta definición de punto borroso nos dificulta mucho su manejo en el momento de la implementación. Por ello, la representación que se ha elegido para modelar el comportamiento de este tipo de puntos es la siguiente.
Cada punto se ve como una lista multidimensional o matriz de valores. Ésta matriz tendrá tantas filas como puntos por los que pase la función de pertenencia se quieran manejar, y tantas columnas como dimensiones tenga el espacio en el que se encuentra el punto.

→ Ejemplo: supuesto que se tiene el siguiente punto borroso, en un espacio bidimensional:



Teniendo en cuenta que quiere representar el número con X valores, la representación que modela la clase punto quedaría de la siguiente manera:



Donde la primera columna representa las coordenadas X y la segunda columna las coordenadas Y.
Estos son los puntos por los que pasa la curva de regresión, por tanto se tiene de una manera implícita la representación del punto borroso.

Cuanto mayor sea la longitud de la lista (mayor número de puntos que representen el número borroso), el punto borroso quedará definido con mayor exactitud, pero por otro lado, como finalmente los cálculos se reducen a operaciones entre listas, el tiempo de ejecución de los algoritmos depende directamente de la longitud de dicha lista, por lo que es posible que con una lista de valores muy larga, el tiempo de resolución tienda a elevarse en exceso. Por esto es necesario encontrar un compromiso entre la longitud de la lista y la exactitud de la definición del número, para qué los valores sean suficientes como para definir el comportamiento del número, y a su vez no se eleve en exceso el tiempo de ejecución.

La elección de esta representación ha sido un acierto en el proceso de implementación, gracias a las facilidades que ha proporcionado en el manejo de éste tipo de números, que a priori presentaba bastante dificultad.

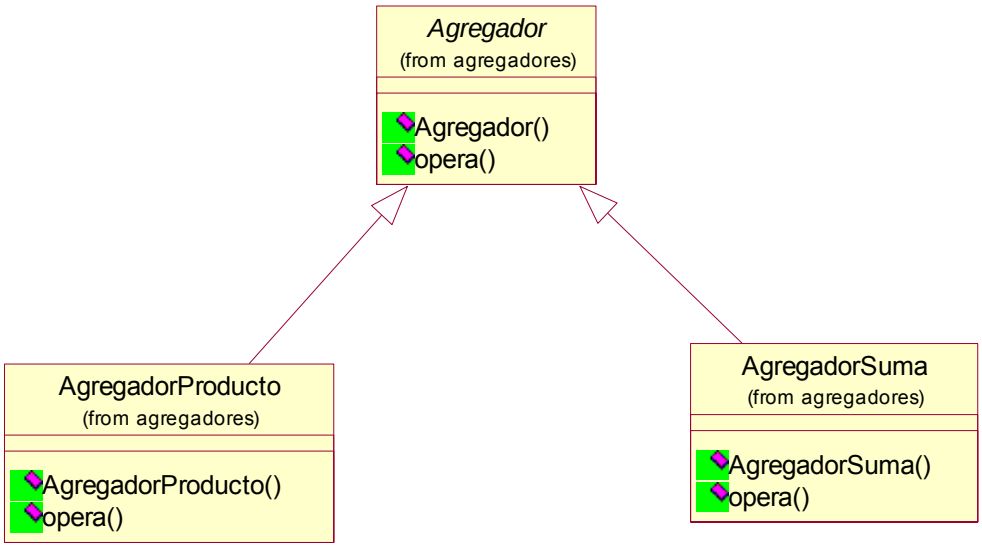
PuntosVarios
Esta clase se encarga de almacenar como constantes diferentes puntos borrosos. Su principal función consiste en proporcionar los elementos a la lista

de parámetros de una curva, en el caso de que el usuario haya decidido que los parámetros de la curva los cree el sistema por defecto.

Al tener una clase solo con constantes, la creación los parámetros por defecto es mucho más sencilla y clara. Basta referenciar tantas constantes cómo parámetros tenga la curva. Además gracias a esta división modular, ante cualquier cambio que se desee realizar en los parámetros iniciales de la curva basta con modificar las constantes ya creadas o crear nuevas dentro de esta clase.

6.3.3.2. Agregadores

Diagrama de clases:



La clase Agregadores es una clase abstracta, cuyas subclases son los diferentes tipos de agregadores que posee el sistema para operar entre números borrosos.
El método principal de la clase es *opera*. Recibe por parámetro dos objetos de tipo Punto (dos punto borrosos) y devuelve otro objeto de tipo Punto.

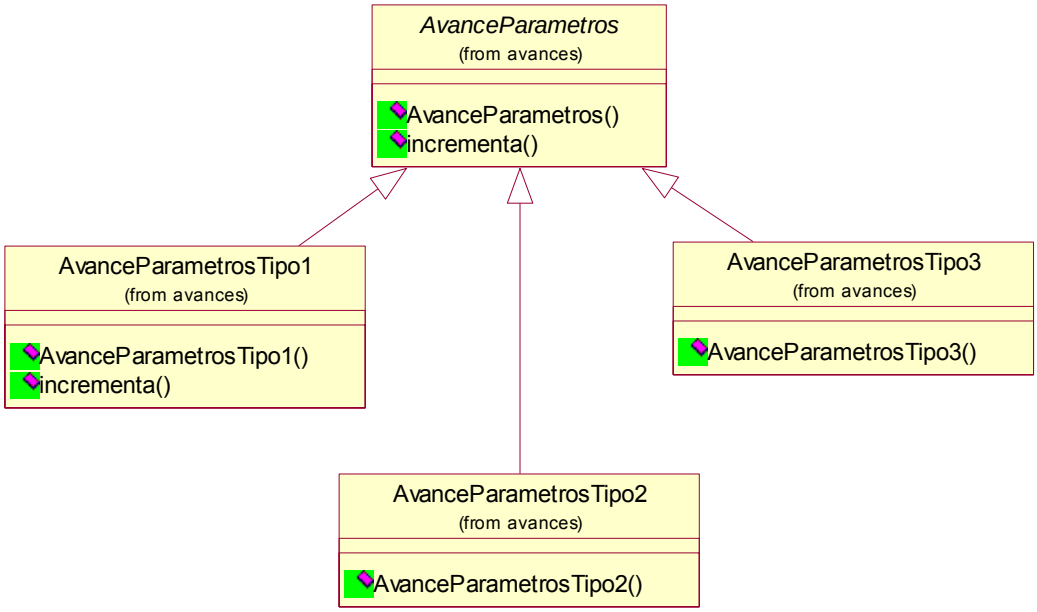
En el caso de que surja la necesidad de usar otros agregadores aparte de los ya implementados, el modo de hacerlo es muy sencillo. Basta con añadir otra subclase a la clase abstracta Agregadores, e implementar el método *opera* con la definición de la operación que va a realizar el nuevo agregador.

Para este prototipo se ha implementado las siguientes subclases
- AgregadorProducto : Su función es calcular el producto entre dos objetos de tipo Punto (dos número borrosos).

- AgregadorSuma : Su función es calcular el producto entre dos objetos de tipo Punto (dos número borrosos).

6.3.3.3. Avances parámetros

Diagrama de clases.



Como se explica en apartados anteriores, la idea principal del algoritmo de regresión que se ha implementado consiste la variación de los parámetros de la curva hasta conseguir unos que se ajusten lo más posible a los puntos que el usuario ha proporcionado al sistema.

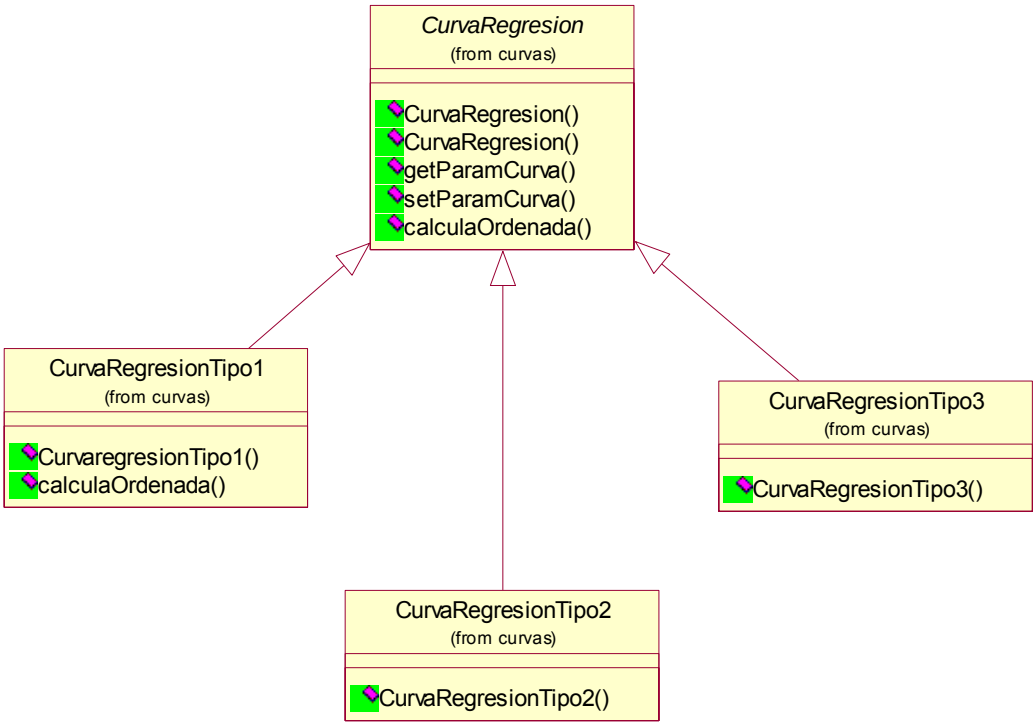
La clase AvanceParametros es abstracta, y cada subclase que implementa su método principal *incrementa*, se corresponde con un algoritmo diferente para llevar a cabo dicha variación.
El usuario decide que algoritmo de variación de parámetros se va a utilizar para llevar a cabo la regresión.

Gracias al diseño modular que se ha llevado a cabo para el desarrollo del sistema, agregar nuevos algoritmos que hagan la manera de variar los parámetros con mayor eficiencia es muy sencillo. Basta con agregar una subclase a la clase AvanceParametros que implemente el método *incrementa* con el nuevo algoritmo.

Para este prototipo se ha implementado la subclase AvanceParametrosTipo1, su metodo incrementa suma uno a cada componente del objeto de tipo Punto que se le pasa por parámetro.

6.3.3.4. Curva regresión.

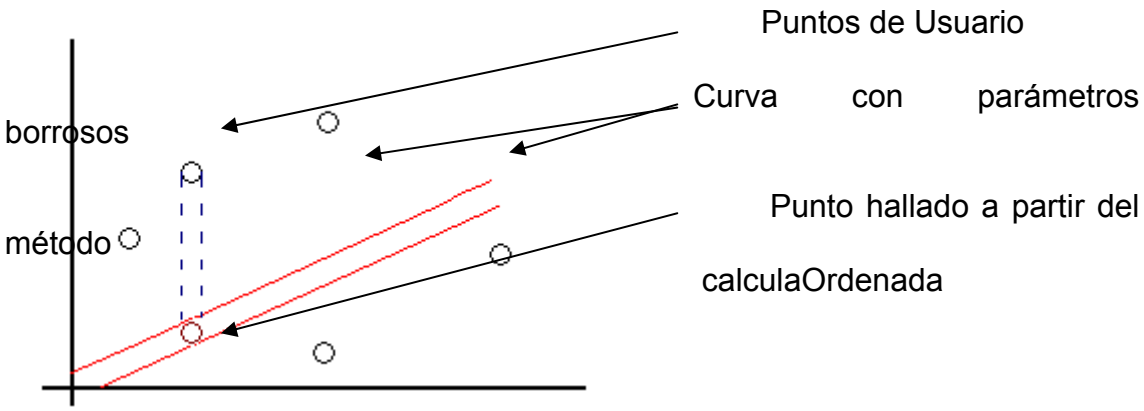
Diagrama de clases.



Esta clase representa la estructura que modela el comportamiento de las curvas en la regresión.
El usuario elegirá la curva que desee, y en función de esto, se creará una curva con un atributo de tipo Parámetros que contendrá una lista parámetros cuyo contenido depende del tipo de curva que se haya elegido.

La clase Curva es una clase abstracta y de ella heredan el resto de curvas que el sistema ofrece para llevar a cabo la regresión.

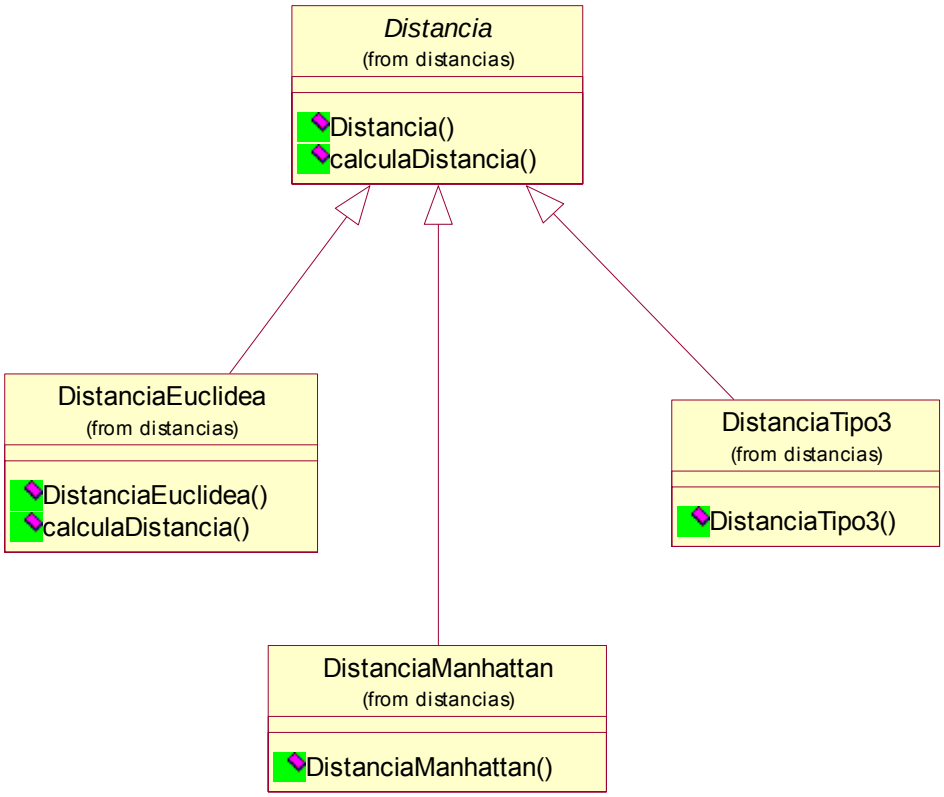
El método más significativo de la clase es calculaOrdenada. Recibe por parámetro las abscisas de un objeto de tipo Punto (ver clase Punto) y devuelve las ordenadas correspondientes a ese punto aplicando la ecuación de la curva.
Éste método se utiliza en la clase motor para saber él punto sobre el cual, se va a calcular la distancia a la curva.
Para saber a qué distancia se encuentra un punto introducido en el sistema por el usuario (P.U.) de la curva, se necesita obtener un punto perteneciente a la curva que se corresponda en abscisas con P.U., y las ordenadas sean las resultantes de la aplicación de la ecuación de la curva.



Para este prototipo se ha implementado la subclase CurvaRegresionTipo1 que representa a la curva Y = aX + b. En su lista de parámetros lleva almacenado el valor del parámetro a y del parámetro b.

6.3.3.5. Distancias

Diagrama de clases:

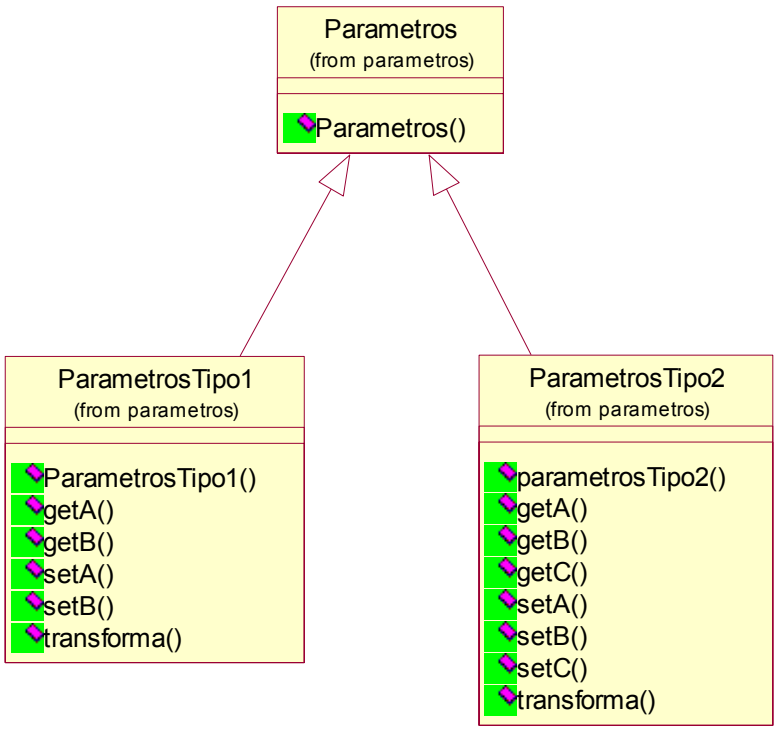


La clase distancia sigue la misma línea de diseño que las clases anteriores. Se trata de nuevo de una clase abstracta. De ella heredan tantas subclases como tipos de distancia requiera el sistema.
En esta aproximación de la implementación de la herramienta únicamente está implementada el cálculo de distancia euclidea entre dos puntos. Este cálculo se lleva a cabo en el método calculaDistancia, que deben implementar todas las subclases de la clase abstracta Distancia.

La distancia del P.U. al punto hallado en la curva (ver definición de clase curva) pertenece también al conjunto de los números borrosos. Por esto, se ha implementado dentro de la clase distancia el métdo calculaDistancia. Éste método recibe por parámetro dos objetos de tipo Punto, calcula su distancia en función de la distancia que desea el usuario y devuelve un objeto de tipo Punto (la distancia entre dos puntos borrosos es un punto borroso). Se puede ampliar la implementación de las distancias que calcula el sistema, añadiendo más tipos de distancia. Gracias al diseño que se ha llevado a cabo, añadir otro tipo de distancia no presenta ninguna dificultad. Basta con añadir otra subclase más a la clase abstracta Distancia, e implementar el método calculaDistancia con el algoritmo correspondiente al nuevo tipo de distancia que se quiere añadir.

6.3.3.6. Parámetros.

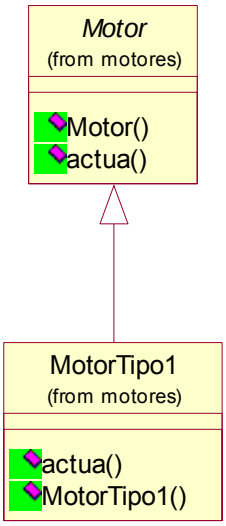
Diagrama de clases.



Esta clase representa los parámetros de cada curva.

La idea principal de realizar una regresión borrosa reside en que los parámetros de cada curva son números borrosos. Además dependiendo del tipo de curva (y por tanto de regresión), ésta necesita resolver una cantidad inicialmente desconocida de parámetros, es el usuario el que va a decidir a que tipo de curva quiere ajustar la regresión.
Por ello, la clase Parámetros almacena como atributo una lista que se gestiona de forma dinámica. Ésta lista esta compuesta por elementos de tipo PuntoMotor ya que los parámetros son borrosos, que inicialmente esta vacía, pero en función del tipo de curva que el usuario elija, la longitud de la lista se adecuará a la curva.
De esta manera, cada curva tiene un objeto de tipo Parámetros, que ya tiene almacenados todos los parámetros de dicha curva, sin tener que gestionarlos ella misma.

6.3.3.7. Motor



La clase Motor es la clase principal de la herramienta. Se trata de una clase abstracta de la que heredan tantas subclases como motores de regresión haya, es decir, tantas como tipos de regresión sea capaz de hacer el sistema.

Para llevar a cabo la regresión, el motor se sirve de todos los elementos anteriormente descritos, opera con ellos y como resultado, obtiene los parámetros de la curva que más se ajustan a los puntos introducidos por el usuario.

Una de las principales características de la herramienta, es que al ser completamente abierta, permitirá en el futuro agregar diferentes motores que realicen regresiones más complejas, gracias a que cada motor de regresión es una subclase de la clase abstracta Motor que se implementa de forma independiente. Ésta característica hace que la herramienta en un futuro pueda llegar a resolver regresiones de todo tipo, para un número indefinido de curvas.

En este prototipo se ha implementado el motor para efectuar regresiones lineales con números borrosos para la curva Y = aX + b.

Para llevar a cabo la regresión al método *actua* (que es el método que calcula la regresión) se le pasa por parámetro los valores que el usuario ha elegido previamente en la interfaz (ver interfaz), es decir, el tipo de curva, el tipo de incrementador de parámetros y el tipo de distancia.

El método actua cuenta con varias listas locales:

- puntosUsuario: Lista que simula los puntos introducidos por el usuario. Éstos son los puntos sobre los que se hará la regresión.
- Distancias: Lista que almacena las distancias de los puntos a la curva. Sus elementos son de tipo Punto (ver explicación clase Distancia).
- parametrosCalc: Lista en la que se almacenan los diferentes parámetros que se van generando. Es necesario almacenar los parámetros porque, a priori no se sabe cuales son los que más se ajustan a los puntos. La decisión de qué parámetros son los apropiados depende de las distancias a la curva que se generen a partir de éstos. Los parámetros que minimicen estas distancias serán los que definan la curva final que más se aproxima a los puntos.

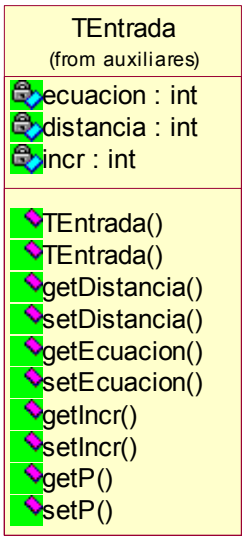
El algoritmo en pseudocódigo:

```
mientras (se pueda continuar incrementando parámetro 1) hacer
        mientras (se pueda continuar incrementando parámetro 2) hacer
                para todos los elementos de puntosUsuario
                        puntoCurva = calcularPuntoCurva (puntoUsuario(i))
                        distancia = calcularDistancia(puntoUsuario(i), puntoCurva)
                        añadir (distancia, distancias)
                        añadir(parametro1,parametro2,parametrosCalc)
                fin para
                incrementa parametro 2
        fin mientras
        incrementa parametro 1
fin mientras
devolver calcularMejorDistancia(distancias)
```

La idea del algoritmo es, mientras no se haya llegado al numero máximo de variaciones de parámetros, se recorre la lista de puntos de usuario y se va calculando y almacenando en la lista de distancias, la distancia entre cada punto y la curva.
Cuando ya se han almacenado todas las distancias para unos parámetros fijos, éstos se varían y se vuelve a repetir el proceso del cálculo de las distancias.
Una vez que ya están todas calculadas, éstas se evalúan para ver cual es la distancia mínima y se devuelven los parámetros que la han generado.

6.3.3.8. Auxiliares

Diagrama de clases.



Ver explicación en apartado de Implementación

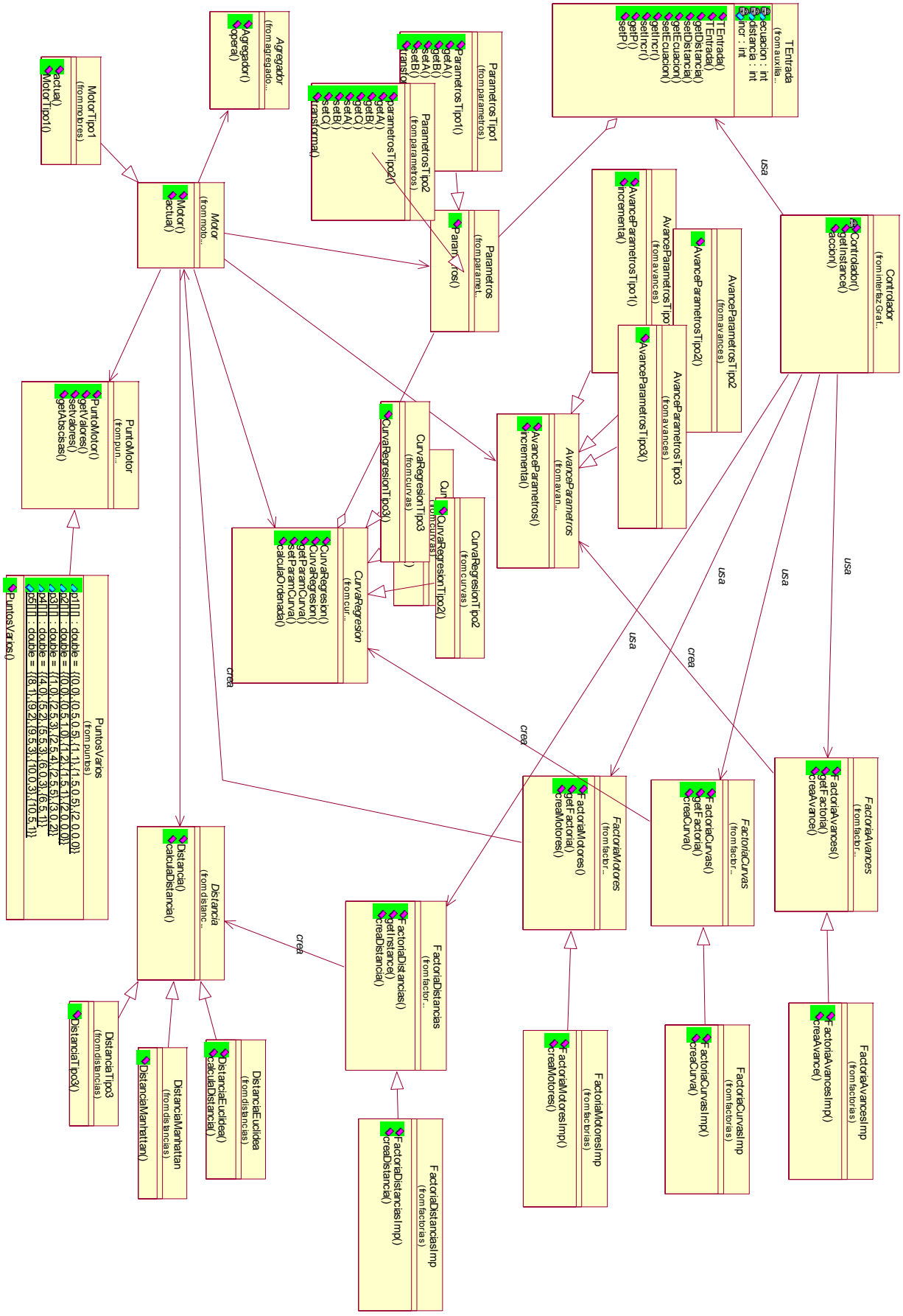
6.3.3.9. Factorías

Diagrama de clases.

Ver explicación en apartado de Implementación.

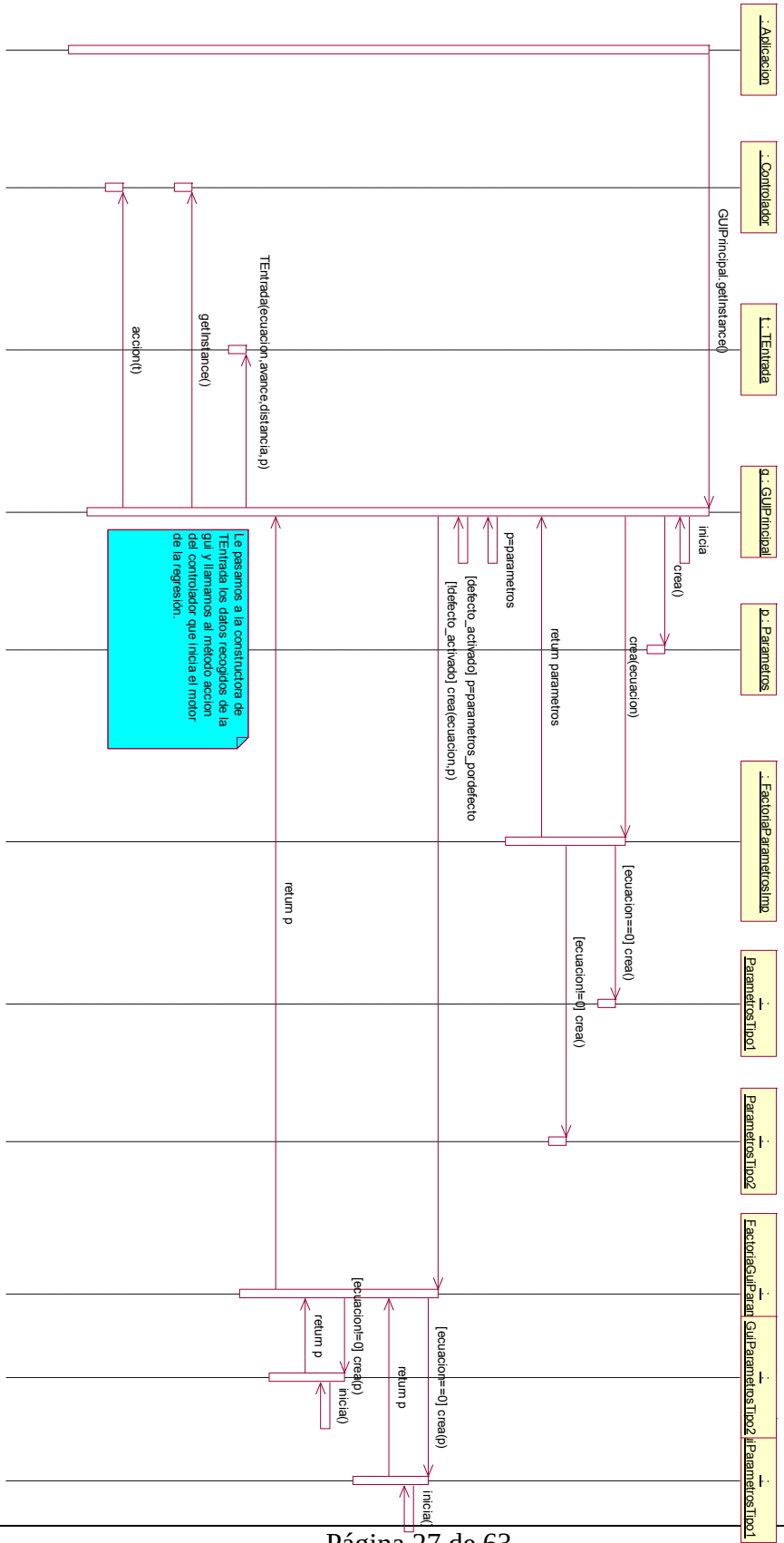
6.3.4. Capa Lógica

A continuación mostramos el diagrama de clases para la capa de lógica:


TEntrada
(from auxilia...
ecuacion : int
distancia : int
incr : int
TEntrada()
TEntrada()
getDistancia()
setDistancia()
getEcuacion()
setEcuacion()
getIncr()
setIncr()
getP()
setP()
Agregador
(from agregado...
Agregador()
opera()
ParametrosTipo1
(from parametros)
ParametrosTipo1()
getA()
getB()
setA()
setB()
transforma()
ParametrosTipo2
(from parametros)
parametrosTipo2()
getA()
getB()
getC()
setA()
setB()
setC()
transforma()
MotorTipo1
(from motores)
actua()
MotorTipo1()
Motor
(from moto...
Motor()
actua()
Parametros
(from parametros)
Parametros()
Controlador
(from interfazGrafi...
getControlador()
getInstance()
accion()
AvanceParametrosTipo1
(from avances)
AvanceParametrosTipo1()
incrementa()
AvanceParametrosTipo2
(from avances)
AvanceParametrosTipo2()
AvanceParametrosTipo3
(from avances)
AvanceParametrosTipo3()
AvanceParametros
(from avan...
AvanceParametros()
incrementa()
PuntoMotor
(from pun...
PuntoMotor()
getValores()
setValores()
getAbscisas()
CurvaRegresionTipo3
(from curvas)
CurvaRegresionTipo3()
CurvaRegresionTipo2
(from curvas)
CurvaRegresionTipo2()
CurvaRegresion
(from cur...
CurvaRegresion()
CurvaRegresion()
getParamCurva()
setParamCurva()
calculaOrdenada()
PuntosVarios
(from puntos)
PuntosVarios()
p1[1] : double = {0.0,0.5,0.5},{1,1},{1.5,0.5},{2,0.0,0}
p2[1] : double = {0.0,0.5,1.0},{1.2},{1.5,1},{2,0.0,0}
p3[1] : double = {1,1,0.2,5.3},{2.5,4},{2.5,5},{3,0.2}
p4[1] : double = {4.0,5.2},{5.5,3.},{6.0,3},{6.5,1}
p5[1] : double = {7.8,1},{9.2},{9.5,3},{10.0,3},{10.5,1}
Distancia
(from distanc...
Distancia()
calculaDistancia()
DistanciaTipo3
(from distancias)
DistanciaTipo3()
DistanciaManhattan
(from distancias)
DistanciaManhattan()
DistanciaEuclidea
(from distancias)
DistanciaEuclidea()
calculaDistancia()
FactoriaDistancias
(from factor...
FactoriaDistancias()
getInstance()
creaDistancia()
FactoriaDistanciasImp
(from factorias)
FactoriaDistanciasImp()
FactoriaMotores
(from factor...
FactoriaMotores()
getFactoria()
creaMotores()
FactoriaMotoresImp
(from factorias)
FactoriaMotoresImp()
creaMotores()
FactoriaCurvas
(from factor...
FactoriaCurvas()
getFactoria()
creaCurva()
FactoriaCurvasImp
(from factorias)
FactoriaCurvasImp()
creaCurva()
FactoriaAvances
(from factor...
FactoriaAvances()
getFactoria()
creaAvance()
FactoriaAvancesImp
(from factorias)
FactoriaAvancesImp()
creaAvance()
usa
usa
usa
usa
usa
crea
crea
crea
crea

6.3.5. Diagramas de Secuencia

- Capa de presentación. Donde interactúan todas las clases relacionadas con la interfaz gráfica.



Explicación del diagrama:

El método main para poder ejecutar el proyecto se encuentra en la clase aplicación.

Aplicación crea una instancia de la gui principal haciéndola visible.

GuiPrincipal ejecuta su método inicia, está a la espera de que se pulsen los botones necesarios para hacer la regresión correctamente (En el apéndice A se muestra el funcionamiento de la interfaz de usuario).

Dependiendo de la ecuación elegida se crearán unos parámetros de un tipo u otro.

Por ejemplo, si elegimos la ecuación tipo 1 se crearán parámetros de tipo 1 puesto que esta clase sólo tiene como atributos el punto borroso a y el punto borroso b.

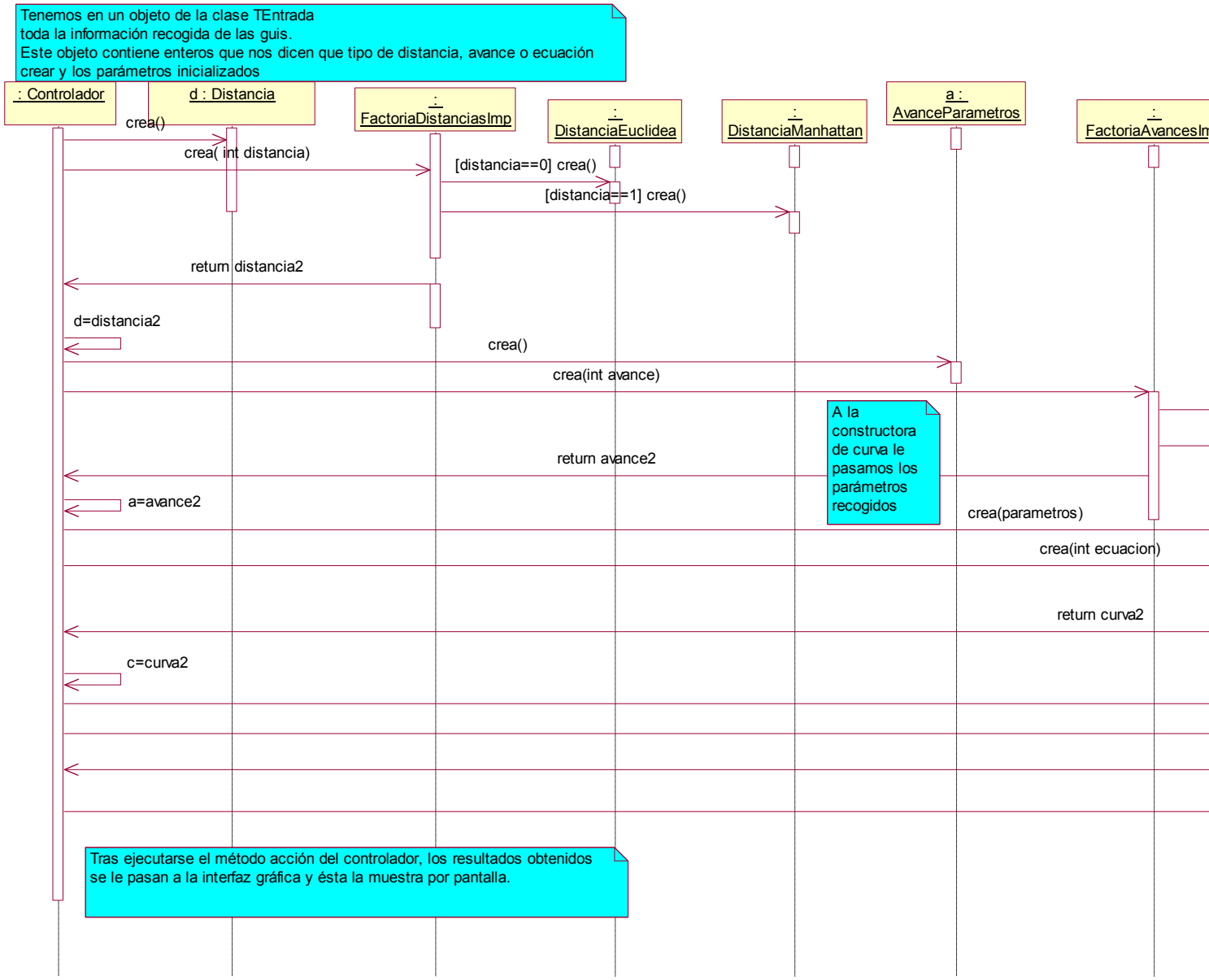
A continuación chequea si el usuario quiere los valores por defecto, y en ese caso inicia el valor de los parámetros a los establecidos por el programa. En caso de que el usuario prefiera introducirlos manualmente se muestra la gui necesaria para escribir los valores deseados.

Todos los datos recogidos (tipo de distancia, tipo de avance, la ecuación y los parámetros iniciales) se pasan a la constructora de la clase de tipo transfer: TEntrada.

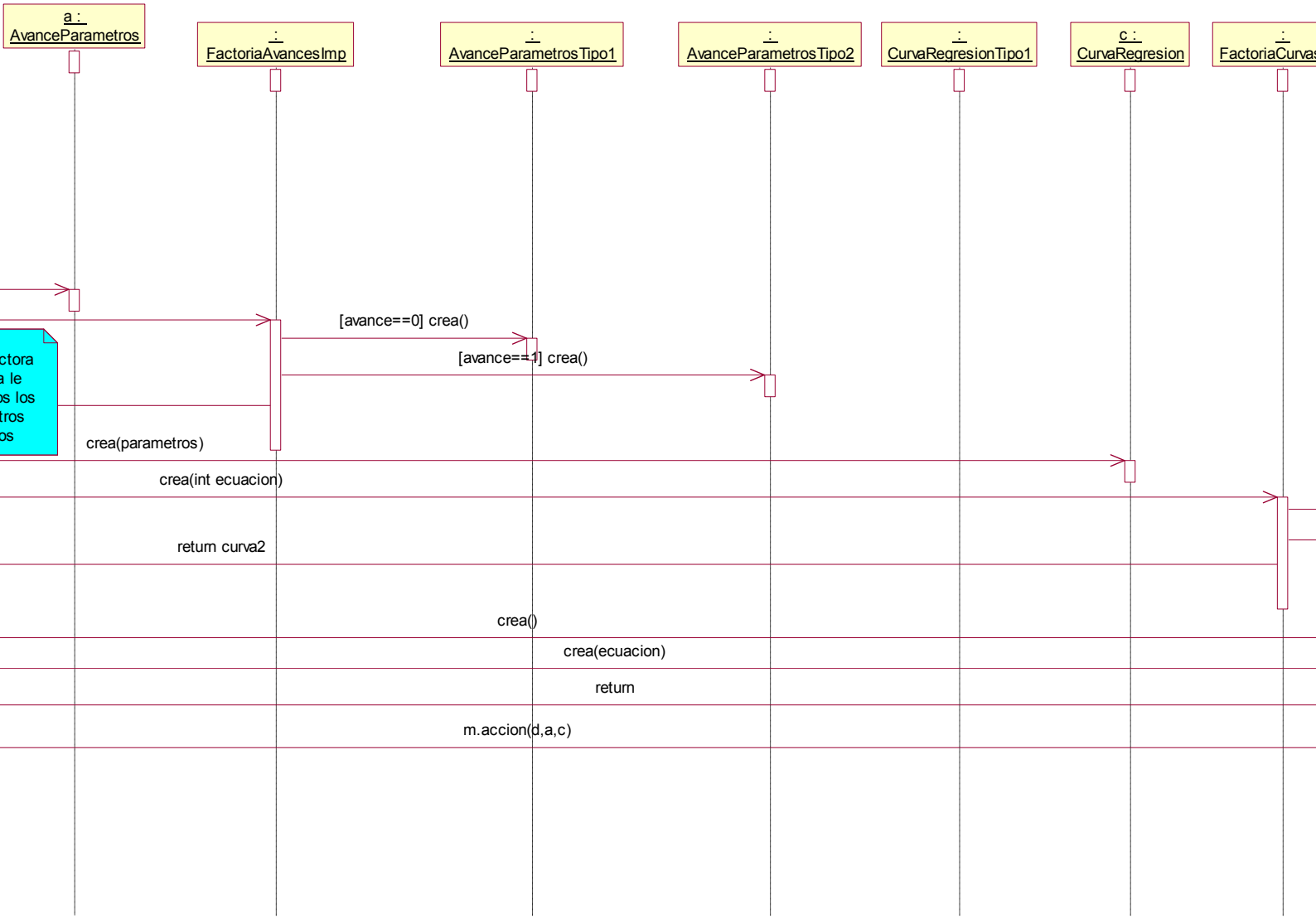
El objeto creado de tipo TEntrada se le pasa al controlador para que ejecute su método acción.

- Interacción entre las clases de la capa lógica

Se muestra el diagrama en dos páginas.

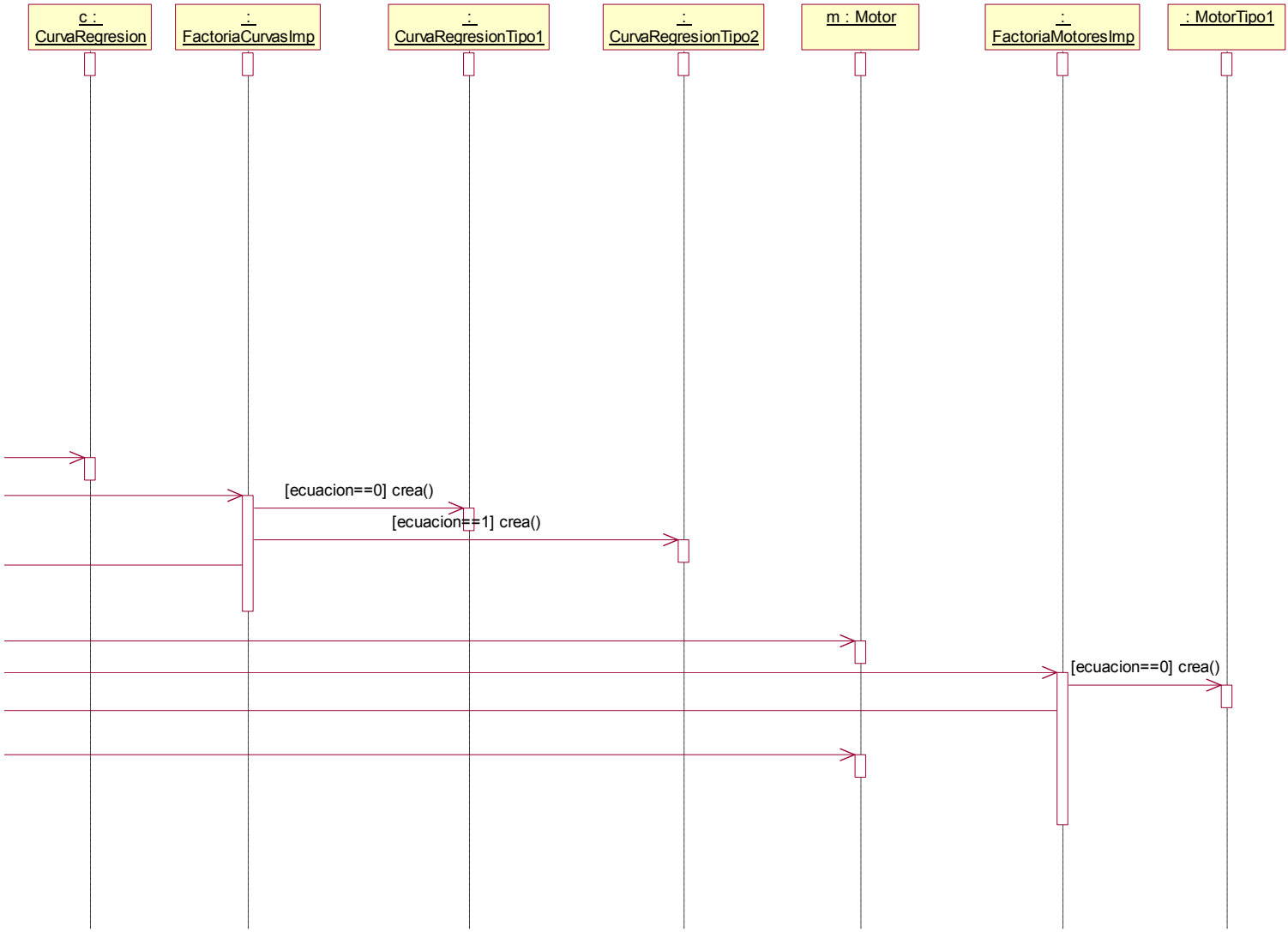


En primer lugar el controlador crea el objeto de tipo distancia. Llama a la factoría encargada de crear las distancias y le pasa un index proveniente del objeto TEntrada que indica el tipo de distancia que hay que crear.



En segundo lugar se crea el objeto de tipo avance. Llama a la factoría encargada de crear los objetos de tipo avance y le pasa un index proveniente del objeto TEntrada que indica el tipo de avance que hay que crear.

En tercer lugar se crea el objeto de tipo curva. Llama a la factoría encargada de crear los objetos de tipo curva y le pasa un index proveniente del objeto TEntrada que indica el tipo de curva que hay que crear.



El método acción del motor se encuentra explicado detalladamente en el apartado 3.3.3.7

6.4. Implementación

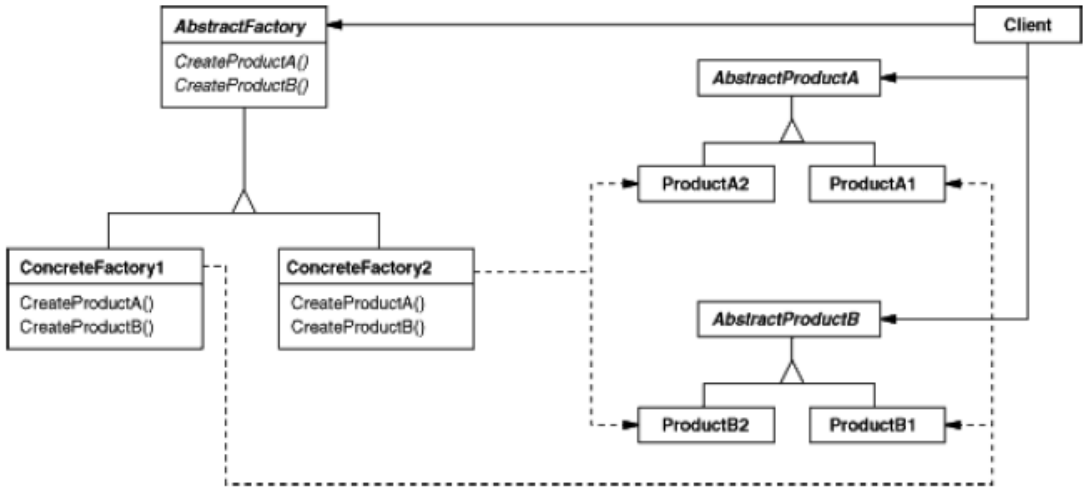
En este bloque se pasa a código lo diseñado. Es en este capítulo donde cobran importancia los patrones de diseño.

A continuación se enumeran los patrones utilizados, y ejemplos de su utilización.

1.Factoría abstracta.

El propósito de este patrón es definir una interfaz para la creación de distintos tipos de objetos relacionados sin necesidad de especificar a qué clase concreta pertenecen.

Estructura:



AbstractFactory define la interfaz para la creación de los productos y las ConcreteFactory implementan dicha interfaz.

AbstractProduct declara la interfaz para un cierto tipo de producto y los ConcreteProduct implementan dicha interfaz y definen un tipo de producto que será creado por la correspondiente ConcreteFactory.

Los clientes tan solo dependen de las interfaces AbstractFactory y AbstractProduct.

Las ventajas que supone utilizar este patrón son las siguientes:

- Aísla a los clientes de las implementaciones concretas de los productos ya que sólo conocen sus interfaces.
- Facilita cambiar una familia de productos por otra, permitiendo configurar un sistema con una de entre varias familias de productos.
- Promueve la consistencia entre los productos.
- Dificulta la definición de nuevo productos.

En nuestro proyecto podemos rescatar los siguientes ejemplos de factorías abstractas:

Podemos elegir entre distintos tipos de ecuaciones para hacer la regresión. Cada ecuación tiene un número determinado de parámetros, por tanto las GUIs necesarias para introducirlos son distintas según la ecuación que se haya elegido.

Necesitamos que la gui que se muestre para la intrducción de parámetros sea seleccionada en tiempo de ejecución. Esto se consigue con la creación de las clases:

FactoriaGuiParametros, FactoriaGuiParametrosImp, GuiParametros, GuiParametrosTipo1, GuiParametrosTipo2...

GuiParametros es la clase padre para GuiParametrosTipo1 y GuiParametrosTipo2.

```
public abstract class GuiParametros extends JDialog{
public GuiParametros(){

}
public abstract void inicia();//throws Exception;
public abstract String getParamA();
public abstract String getParamB();
public abstract String getParamC();
public abstract void setParamA(String a);
public abstract void setParamB(String a);
public abstract void setParamC(String a);
public abstract PuntoMotor getPuntoA();
public abstract PuntoMotor getPuntoB();
public abstract PuntoMotor getPuntoC();
//public abstract boolean isCorrectos();
}

public class GuiParametrosTipo1 extends  GuiParametros{
    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private JPanel panelParametros= new JPanel();
    private JPanel panelBotones= new JPanel();
    private GridLayout gridLayout1 = new GridLayout();
                                        *
                                        *
                                        *
*Todo el código referente a la gui de tipo1
                                        *
                                        *
}


public class GuiParametrosTipo2 extends  GuiParametros{
    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
```

```
    private JPanel panelParametros= new JPanel();
    private JPanel panelBotones= new JPanel();
    private GridLayout gridLayout1 = new GridLayout();
                                        *
                                        *
                                        *
*Todo el código referente a la gui de tipo2
                                        *
                                        *
}
```

FactoriaGuiParametros la clase abstracta padre de FactoriaGuiParametrosImp.

```
public abstract class FactoriaGuiParametros {
        private static FactoriaGuiParametros unicaInstancia;
    public FactoriaGuiParametros() {
                }
    public static FactoriaGuiParametros getFactoria(){
    if (unicaInstancia==null)
unicaInstancia = new FactoriaGuiParametrosImp();
    return unicaInstancia;
                }
//              ---------------------------------------------------------------------------
    public abstract GuiParametros creaGui(int tipo);
//              ---------------------------------------------------------------------------                 }
}
```

Para crear la Gui deseada se llama al método crea de FactoriaGuiParametrosImp, pasándole como atributo el tipo de ecuación que es un entero recogido de la Gui principal. Según el valor del atributo se creará una gui de parámetros de un tipo u otro.

```
public class FactoriaGuiParametrosImp extends FactoriaGuiParametros{
    public FactoriaGuiParametrosImp() {
    super();
        }
    public GuiParametros creaGui(int tipo){
    GuiParametros gui;
    switch(tipo){
    case 0:
    gui= GuiParametrosTipo1.getInstance();
    return gui;
    case 1:
    gui= GuiParametrosTipo2.getInstance();
```

```
    return gui;
    case 2:
    gui= GuiParametrosTipo2.getInstance();
    default: gui= GuiParametrosTipo2.getInstance();
        }
    return gui;
}
}
```

Otro ejemplo de utilización de la factoría abstracta está en el método que queramos utilizar para calcular las distancias entre puntos borrosos.

Tenemos las siguientes clases:

FactoriaDistancias. Clase abstracta padre de FactoriaDistanciasImp.

```
public abstract class FactoriaDistancias {
        private static FactoriaDistancias unicaInstancia;
        public FactoriaDistancias() {
super();
// TODO Auto-generated constructor stub
        }
        public static FactoriaDistancias getFactoria(){
if (unicaInstancia==null)
unicaInstancia = new FactoriaDistanciasImp();
    return unicaInstancia;
        }
    public abstract Distancia creaDistancia(int tipo);
}
```

FactoriaDistanciasImp. Clase que crea un tipo de distancia u otra según la elegida en la gui.

```
public class FactoriaDistanciasImp extends FactoriaDistancias{
        public FactoriaDistanciasImp() {
super();
// TODO Auto-generated constructor stub
        }
    public Distancia creaDistancia(int tipo){
    Distancia cur;
    switch(tipo){
    case 0:
    cur= new DistanciaEuclidea();
    return cur;
    case 1:
    cur= new DistanciaManhattan();
    return cur;
```

```
    case 2:
    cur= new DistanciaTipo3();
    return cur;
    default: cur= new DistanciaEuclidea();
    return cur;
                }
        }
}
```

Distancia. Clase padre abstracta para los diferentes tipos de distancias:

```
public abstract class Distancia {

    public Distancia() {
    super();
        }

public abstract PuntoMotor calculaDistancia(PuntoMotor a, PuntoMotor b);
}
public class DistanciaEuclidea extends Distancia {
    public DistanciaEuclidea(){
    super();
        }
public PuntoMotor calculaDistancia (PuntoMotor a, PuntoMotor b) {
    PuntoMotor distancia;
ArrayList <Double> acumDistancia = new ArrayList <Double>();
    double [][] aA = a.getValores();
    double [][] bA = b.getValores();
                                        *
                                        *
                                        *
                                        *
    *Código de  la distancia euclidea
}
```

Cuando queremos crear un tipo distancia llamamos a la factoría de la siguiente forma:

```
Distancia d= FactoriaDistancias.getFactoria().creaDistancia(t.getDistancia());
```

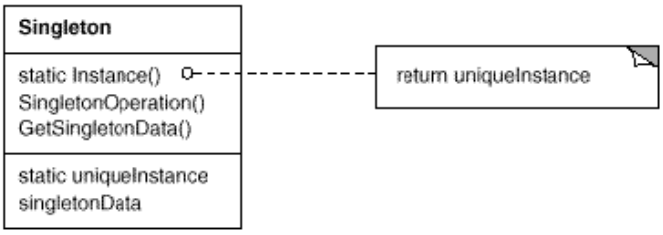
La variable d será de tipo DistanciaEuclidea, DistanciaManhattan o cualquiera de las clases hijas de Distancia.

PATRÓN SINGLETON Propósito:

Asegura que solo existe una instancia de una determinada clase y proporciona la forma de acceder a ella.

Hay muchos recursos únicos en un sistema. En nuestro caso encontramos ejemplos de este patrón en la creación de las factorías o en la creación de las guis que forman la interfaz gráfica.

Estructura:



Singleton define un método de clase que permite acceder a la instancia única y puede ser el responsable de crear dicha instancia.

Ventajas del uso del patrón Singleton:

Acceso controlado a la instancia.
Mantiene limpio el espacio de nombres al evitar el uso de variables globales.
Permite definir subclases de la instancia única y en tiempo de ejecución se puede determinar qué tipo utilizar.
Permite mantener un número variable de instancias.
Es más flexible que conseguir el mismo resultado con métodos de clase:

- permite más de una instancia.
- permite la especialización polimórfica.

Ejemplo de utilización del patrón singleton.

El controlador del sistema debe ser creado una única vez.

```
public class Controlador {
        private static Controlador instance = null;
//      ----------------------------------------------------------------------
    private Controlador() {}
//      ----------------------------------------------------------------------
    public static Controlador getInstance() {
    if (instance == null) instance = new Controlador();
    return instance;
        }
//      ----------------------------------------------------------------------
    public void accion(Object datos){
                                        *
                                        *
                                        *
}
```

Desde la clase principal del sistema creamos la guiPrincipal, usando también el patrón singleton. En el método acción de la gui principal creamos el controlador y le pasamos las opciones elegidas en la interfaz gráfica.

```
public class Aplicacion {
        public Aplicacion() {
GUIPrincipal.setDefaultLookAndFeelDecorated(true);
GUIPrincipal marco = GUIPrincipal.getInstance();
    marco.validate();
                                        *
                                        *
                                        *
                                        *
}
public class GUIPrincipal extends JFrame implements IGUIPrincipal{
                                        *
                                        *

//-----------------------------------------------------------------------
public static GUIPrincipal getInstance(){
        if (instancia == null)
        instancia = new GUIPrincipal();
    return instancia;
}
//-----------------------------------------------------------------------
private GUIPrincipal() {
        try {
setDefaultCloseOperation(EXIT_ON_CLOSE);
inicia();
        }
    catch (Exception exception) {
    exception.printStackTrace();
        }
}
//-----------------------------------------------------------------------
private void inicia() throws Exception {
                *
                                        *
    TEntrada t = new TEntrada(indice,distancia,incr,par);
    Controlador.getInstance().accion(t);
                                        *
                                        *
}
```

La importancia de la clase TEntrada

A través de la gui principal recogemos una serie de informaciones que deben llegar a la vez al motor encargado de la regresión:

El tipo de distancia empleada, el tipo de avance de los parámetros, la ecuación elegida y los parámetros creados (automáticamente por el sistema o manualmente interactuando con el usuario).

Por tanto creamos una clase TEntrada cuya función es almacenar toda esta información y pasarla al controlador para que opere con ella.

Por cada campo creamos un atributo en la clase:

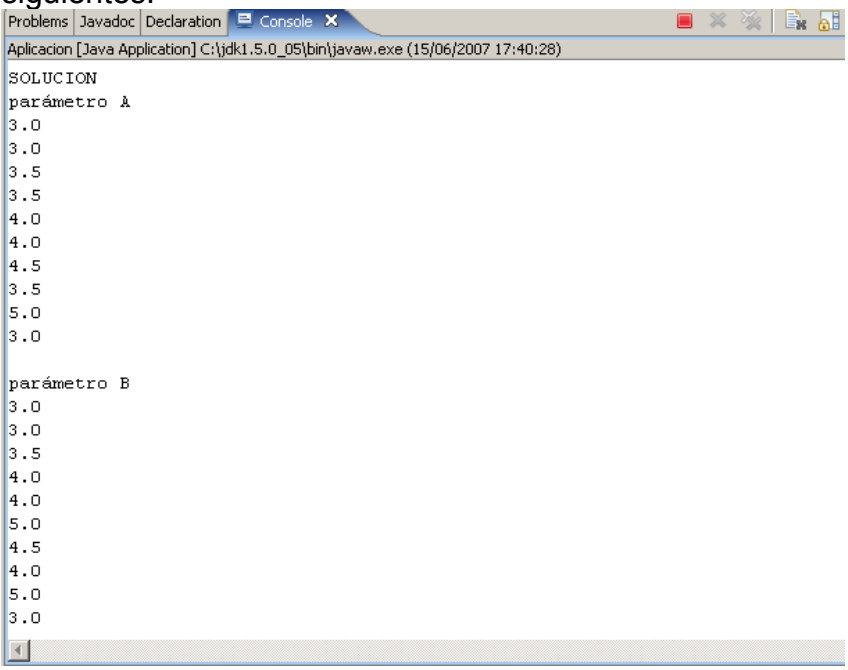
```
public class TEntrada {
private int ecuacion;
private int distancia;
private int incr;
private Parametros p;
public TEntrada() {
    super();
    // TODO Auto-generated constructor stub
}
public TEntrada(int ecuacion, int distancia, int incr, Parametros p) {
    super();
    this.ecuacion = ecuacion;
    this.distancia = distancia;
    this.incr = incr;
    this.p = p;
}
public int getDistancia() {
    return distancia;
}
public void setDistancia(int distancia) {
    this.distancia = distancia;
}
public int getEcuacion() {
    return ecuacion;
}
public void setEcuacion(int ecuacion) {
    this.ecuacion = ecuacion;
}
public int getIncr() {
    return incr;
}
public void setIncr(int incr) {
    this.incr = incr;
}
public Parametros getP() {
    return p;
}
```

```
public void setP(Parametros p) {
    this.p = p;
}
```

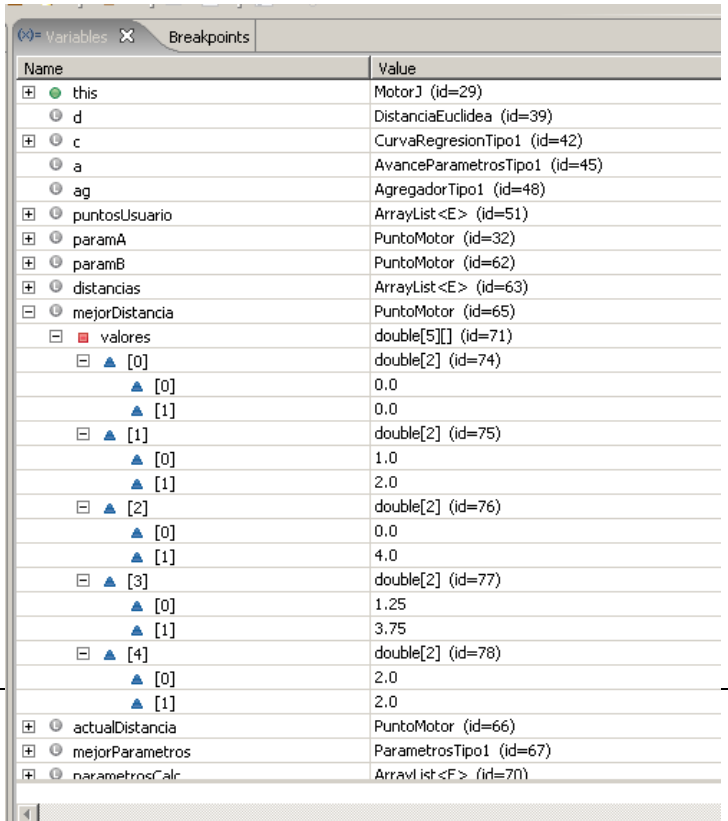
6.5. Pruebas

Realizamos una regresión utilizando la curva y=ax + b, distancia de tipo euclídea, avance de parámetros de tipo 1, parámetros por defecto, agregador de tipo1, y número de dimensiones 2.

El valor de los parámetros para los que la regresión se hace óptima son los siguientes:

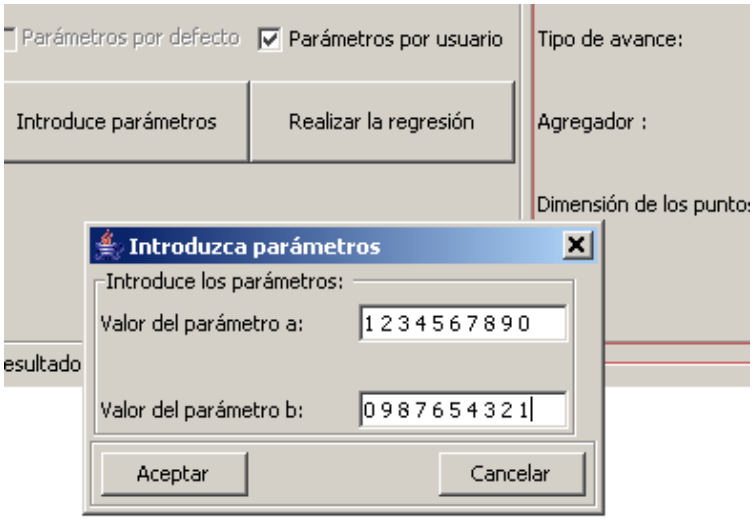


La distancia correspondiente a estos parámetros es la siguiente:

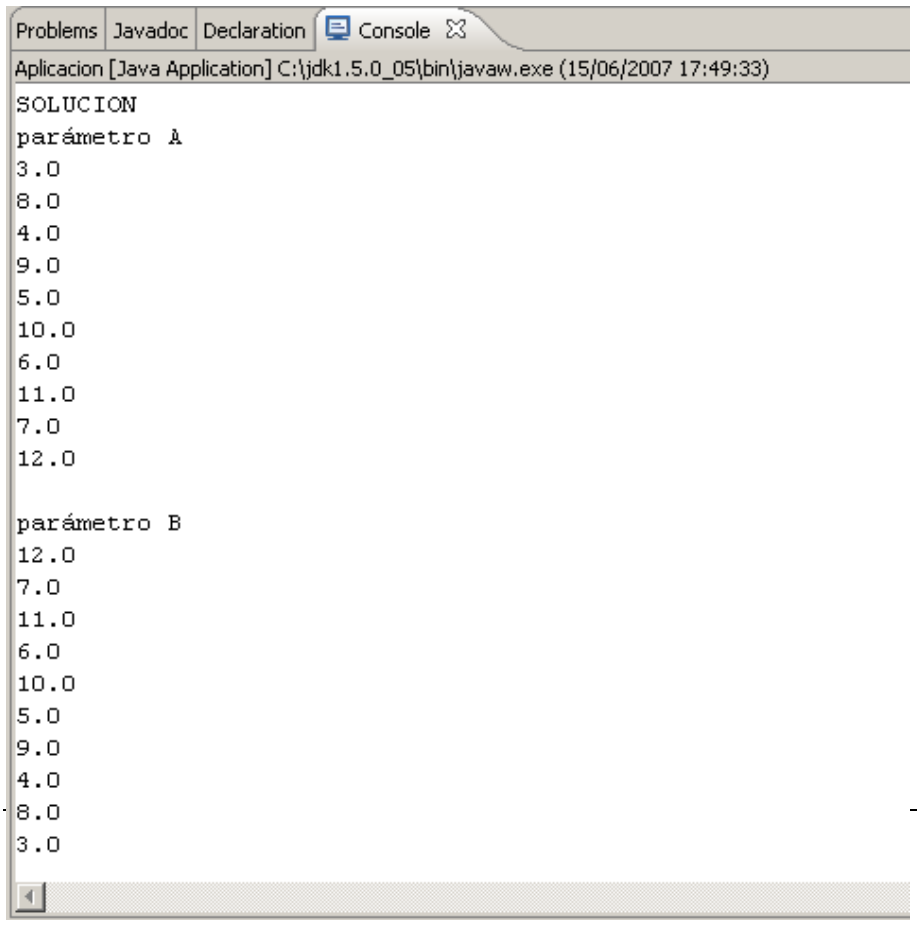


Ahora realizamos la regresión utilizando parámetros introducidos por usuario, utilizando la curva y=ax + b, distancia de tipo euclídea, avance de parámetros de tipo 1, agregador de tipo1, y número de dimensiones 2.

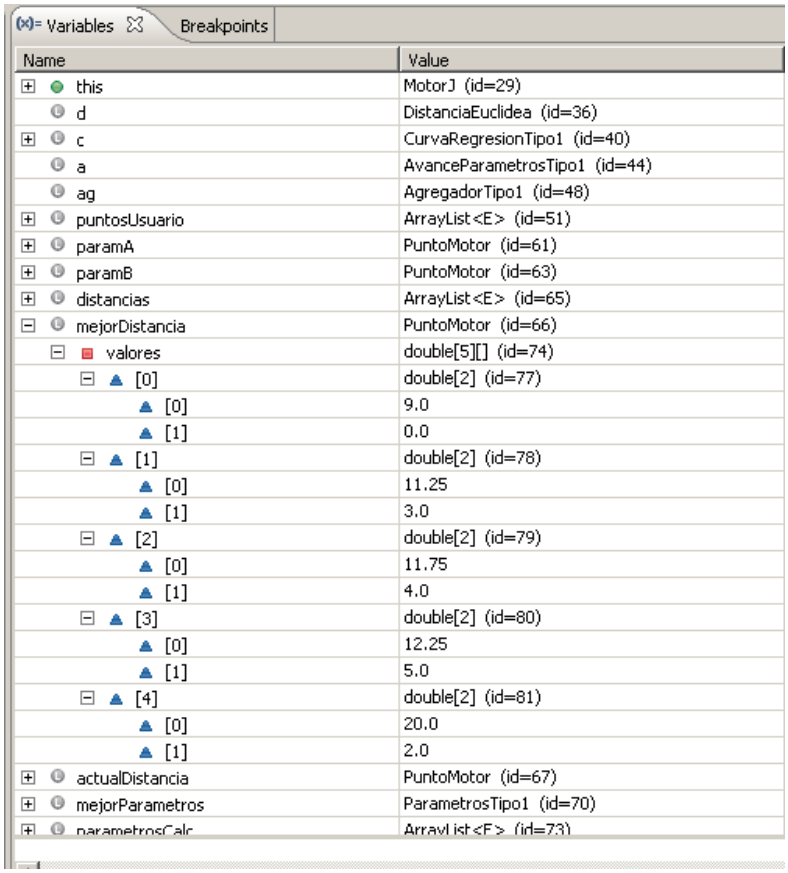
Los parámetros introducidos son los siguientes:



El valor de los parámetros para los que la regresión se hace óptima son los siguientes:



La distancia correspondiente a estos parámetros es la siguiente:



7. Líneas futuras:

7.1. Una posible notación para las ecuaciones no lineales.

Las ecuaciones estarán formadas por terminales y funciones.

Lo terminales pueden ser de dos formas, valores o parámetros. Como valores comprendemos cualquier número escrito en notación entera (12, 456, 7, etc) y el número “e” que se representa como la letra e.
Los parámetros son cualquier letra en minúscula exceptuando la letra e (a, b, c, …, z).
Las funciones pueden ser:
El exponente: se representa como exp(x,y), donde x es la base e “y” el exponente.
La raíz cuadrada: sqrt(x).

El logaritmo en base x: log(x,y), donde x es la base del logaritmo e ”y” el número al que se le aplica.
La suma: +
La multiplicación: *
La resta: -
La división: /

Esta gramática se podrá ampliar. Se deberá crear un compilador que controle la ecuación escrita sea correcta.

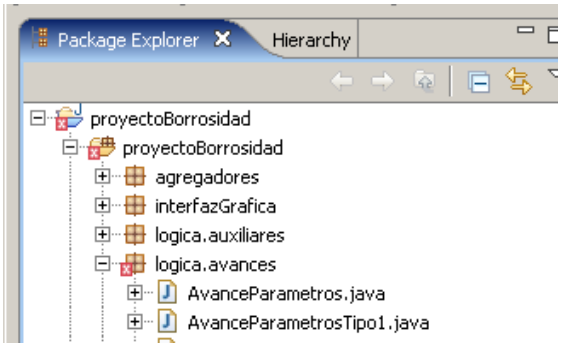
7.2. Introducción de nuevas funcionalidades

Como se ha mencionado anteriormente el diseño del proyecto es abierto de cara a futuras ampliaciones que se quieran hacer en la aplicación.
Como ampliaciones entendemos la introducción de nuevos métodos de cálculo de distancias, nuevos algoritmos para el avance de parámetros, agregadores o ecuaciones para hacer la regresión.

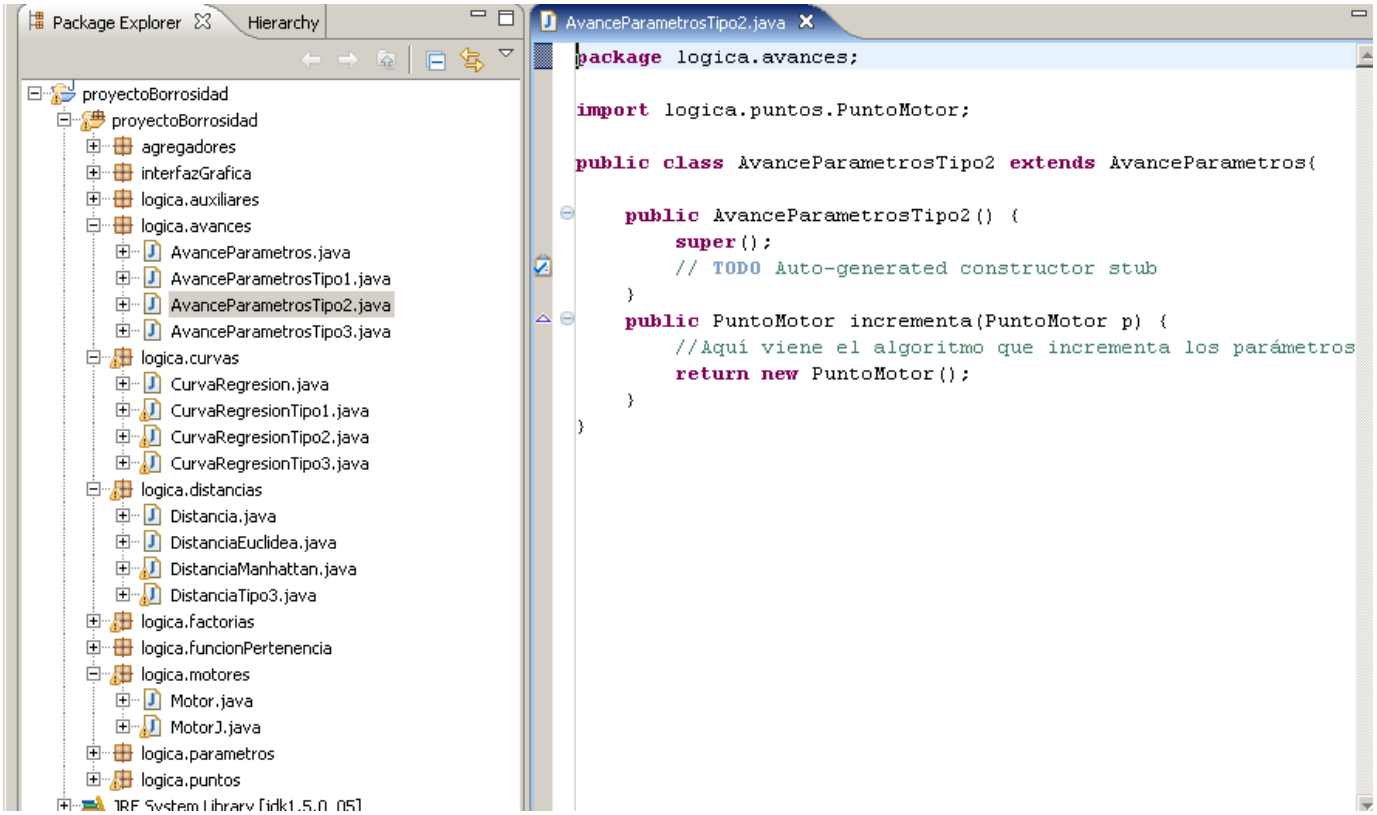
A continuación se muestra un ejemplo de lo automático y nada engorroso que resulta la introducción de un nuevo tipo de avance para los parámetros.

En primer lugar tenemos que crear con anterioridad la clase en la que está implementado el funcionamiento de este tipo de avance.

El siguiente paso es buscar el paquete en el que se agrupan todos los tipos de avances.



Añadimos como nuevo tipo de avance la clase creada anteriormente, indicando que es hija de la clase abstracta AvanceParametros:



Para finalizar modificamos la factoría que se encarga de la creación de avances añadiendo un nuevo caso en el switch de la constructora:

```
public class FactoriaAvancesImp extends FactoriaAvances{

    public FactoriaAvancesImp() {
        super();
    }

    public AvanceParametros creaAvance(int tipo){
        AvanceParametros ava;
        switch(tipo){
        case 0:
            ava= new AvanceParametrosTipo1();
            return ava;
        case 1:
            ava= new AvanceParametrosTipo2();
            return ava;
        case 2:
            ava= new AvanceParametrosTipo3();
            return ava;
        default: ava= new AvanceParametrosTipo1();
        return ava;
        }
    }
}
```

Para que el nuevo tipo de avance aparezca en la interfaz gráfica debemos añadir el nombre del tipo de avance en la lista de Strings que forman el combo de avances de la interfaz gráfica.

```
String[] incr= {"Avance Tipo1","Avance Tipo2","Avance Tipo3"};
comboIncr= new JComboBox(incr);
```

8. Conclusiones

En este apartado se detallan las conclusiones obtenidas después del proceso de análisis, diseño e implementación de la herramienta abierta de regresión usando lógica difusa.

Estudios anteriores avalan que en ocasiones el análisis de regresión borroso produce resultados que se ajustan mejor a los obtenidos mediante el análisis de regresión usando puntos nítidos. Esto se debe a que la naturaleza difusa de los parámetros permite una mejor representación de las observaciones.

Con nuestro proyecto se ha desarrollado una herramienta de regresión borrosa que puede trabajar con imprecisión en la información y capaz de determinar que valores de los parámetros son los óptimos para realizar la regresión. Siguiendo la idea planteada inicialmente sobre elaborar una herramienta abierta, todas las etapas por las que se ha pasado se han cuidado especialmente en este aspecto, teniéndolo presente en todo momento. El proceso de diseño de la herramienta para que fuera completamente abierta supuso un paso adelante en nuestra capacidad de diseño.

También se ha desarrollado una interfaz gráfica sencilla y fácil de utilizar para los futuros usuarios de la aplicación. El usuario podrá elegir entre las diferentes opciones instaladas en la aplicación para la realización de la regresión (modelos de curvas, métodos de cálculo de distancias, etc). A la hora de elegir los elementos que formarán parte del proceso de regresión borrosa, el usuario, dependiendo de sus necesidades, deberá escoger los tipos de elementos más adecuados para llevarlo a cabo,
ya que el resultado que se obtendrá estará determinado por dichos elementos.

Una de las mayores dificultades cara a la implementación del sistema ha sido el enfrentamiento con una gran cantidad de conceptos matemáticos sobre regresiones que pese a haber estudiado y comprendido en cursos pasados, la implementación de un sistema que lo lleve a cabo es mucho más complejo. En particular, ha presentado especial dificultad la representación y elección de estructura de datos para los números borrosos así como la implementación de un algoritmo que determine la mejor configuración de los parámetros, debido al desconocimiento inicial de cualquier noción teórica sobre lógica borrosa.

Hemos seguido un modelo secuencial que nos ha permitido mantener un proceso de desarrollo organizado.

En base al trabajo realizado y a los resultados obtenidos en las pruebas, se demuestra que el uso de la herramienta de regresión borrosa aporta evidencias suficientes de viabilidad.

Con la implementación de esta herramienta se desea establecer una base para futuros trabajos que vayan añadiendo funcionalidades hasta conseguir una herramienta robusta y completa capaz de ser utilizada realmente para facilitar los estudios en los diferentes campos de aplicación de la estadística.

9. Apéndice

9.1. Apendice A : Funcionamiento de la Interfaz Gráfica

El objetivo de nuestro diseño es tener una interfaz gráfica lo más sencilla posible en la que la interacción del usuario con el programa sea mínima.

Debemos diferenciar entre las dos tipos de regresiones (lineales y no lineales) que se pueden realizar, reutilizar las opciones que una necesita para la otra, de tal forma que se ahorra espacio y sobretodo se consigue un efecto que no fatiga la vista del usuario.

La Gui principal toma un aspecto organizado gracias al uso de un BorderLayout, en el que se encuentran diferentes paneles de tipo JPanel.

El borderLayout se divide en en cinco zonas situadas según su nombre indica: NORTH, EAST, WEST, SOUTH y CENTER.

En el panel situado en el norte, están los checkbox que el usuario debe activar dependiendo del tipo de regresión que quiere realizar. A su vez este panel también e organiza con un BorderLayout a diferencia de que no utiliza todas las zonas.

//panel tipo de regresión

panelTipoRegresion.add(tipoRegresion,BorderLayout.NORTH);

panelTipoRegresion.add(jCheckBoxLineal,BorderLayout.WEST);

panelTipoRegresion.add(jCheckBoxNoLineal,BorderLayout.EAST);

En el panel situado en la zona oeste se encuentra todo lo relacionado únicamente con las regresiones lineales. Su organización es de GridLayout: trata el panel como una matriz de m x n elementos.

//panel regresion lineal

```
panelParametrosLineal.add(curvaLineal);
panelParametrosLineal.add(comboCurvas);
panelParametrosLineal.add(porDefecto);
panelParametrosLineal.add(porUser);
panelParametrosLineal.add(botonParametros);
panelParametrosLineal.add(botonEjecutarLineal);
```

El panel central contiene los parámetros que son comunes para los dos tipos de regresiones. Usa organización de Gridlayout:

//panel comun para regresiones lineales y no lineales

```
panelParametrosComun.add(tipoDistancia);
panelParametrosComun.add(comboDistancias);
```

panelParametrosComun.add(tipoIncrementador);

panelParametrosComun.add(comboIncr);

En el panel situado en la zona este se encuentra lo relacionado con las regresiones no lineales:

//panel no lineal

panelParametrosNoLineal.add(curvaNoLineal);

panelParametrosNoLineal.add(botonLinealizarYEjecutar);

Finalmente en la zona sur se enuentra el área que muestra los resultados obtenidos por la aplicación:

//panel resultados

panelResultados.add(areaResultados);

Introducimos los paneles en el borderLayout de la Gui principal:

contentPane.add(panelTipoRegresion,BorderLayout.NORTH);

contentPane.add(panelParametrosLineal,BorderLayout.WEST);

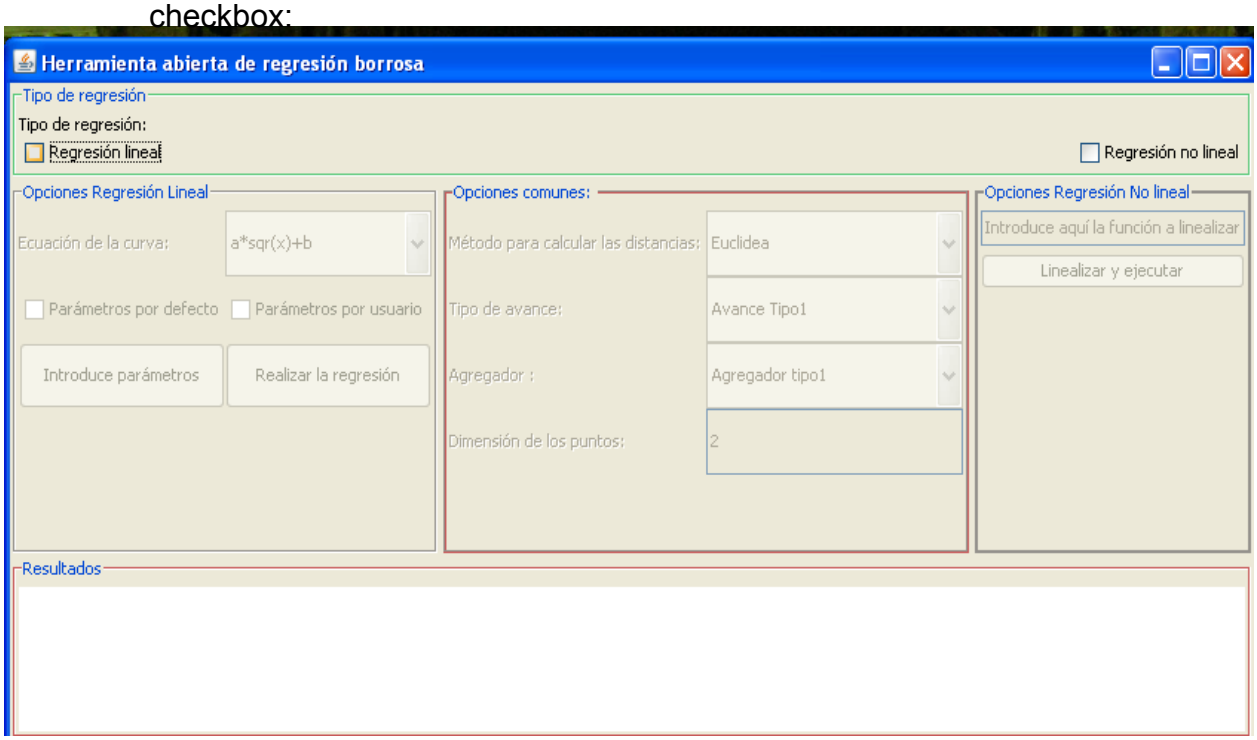
contentPane.add(panelParametrosComun,BorderLayout.CENTER);

contentPane.add(panelParametrosNoLineal,BorderLayout.EAST);

contentPane.add(panelResultados, BorderLayout.SOUTH);

Inicialmente aparece la siguiente interfaz gráfica.

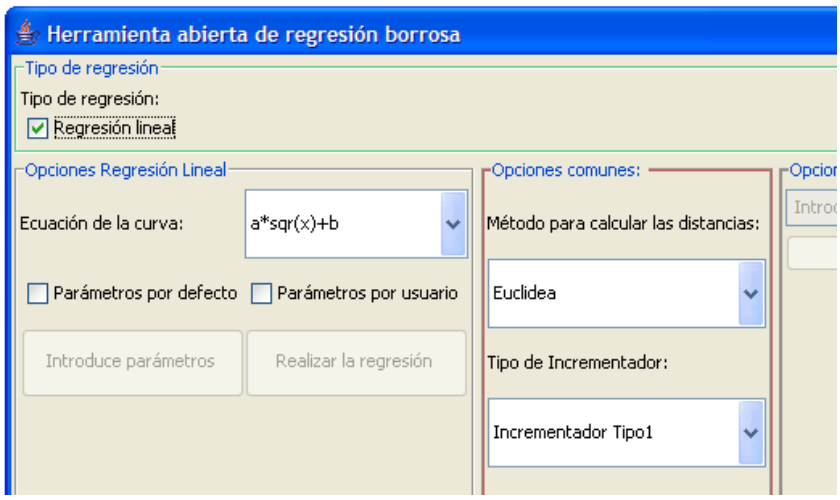
Debemos decidir si lo que queremos hacer es una regresión utilizando ecuaciones lineales o no lineales haciendo click sobre uno de los dos checkbox:



Elegimos regresión lineal.

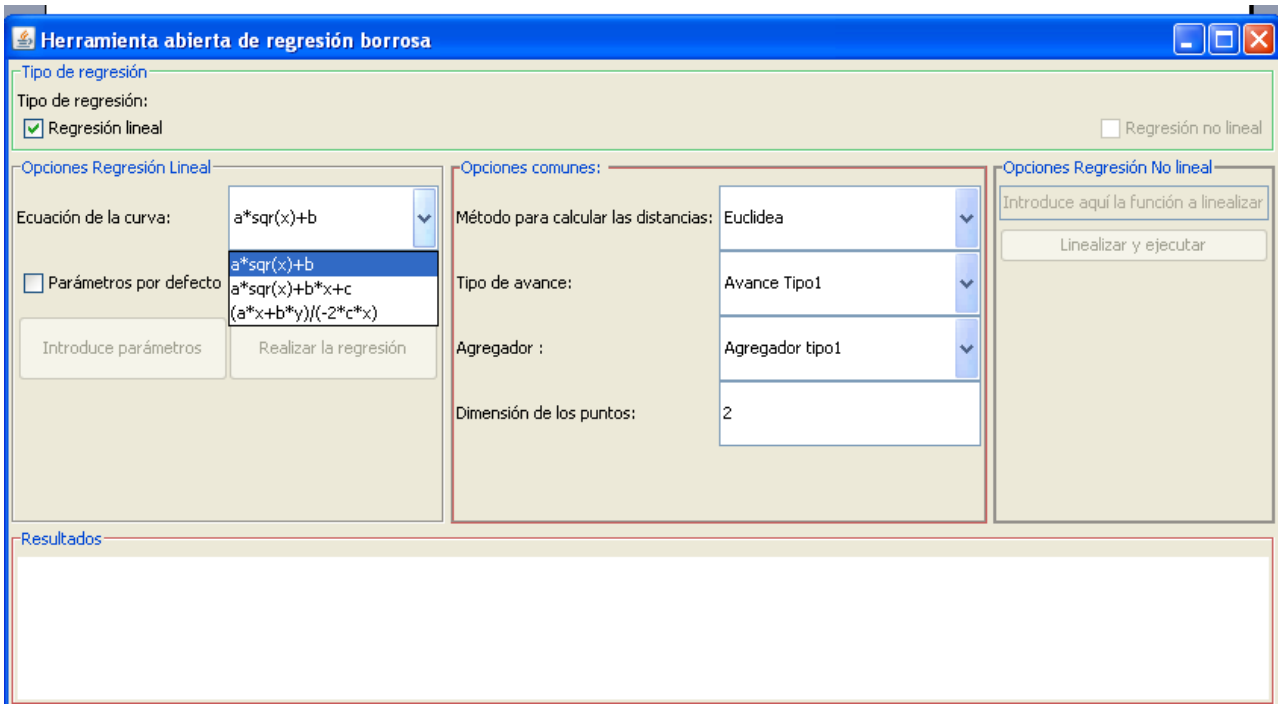
Se habilitan las siguientes opciones:

Ecuación de la curva a elegir, la manera de elegir los parámetros iniciales, el tipo de distancia y el tipo de incrementador.

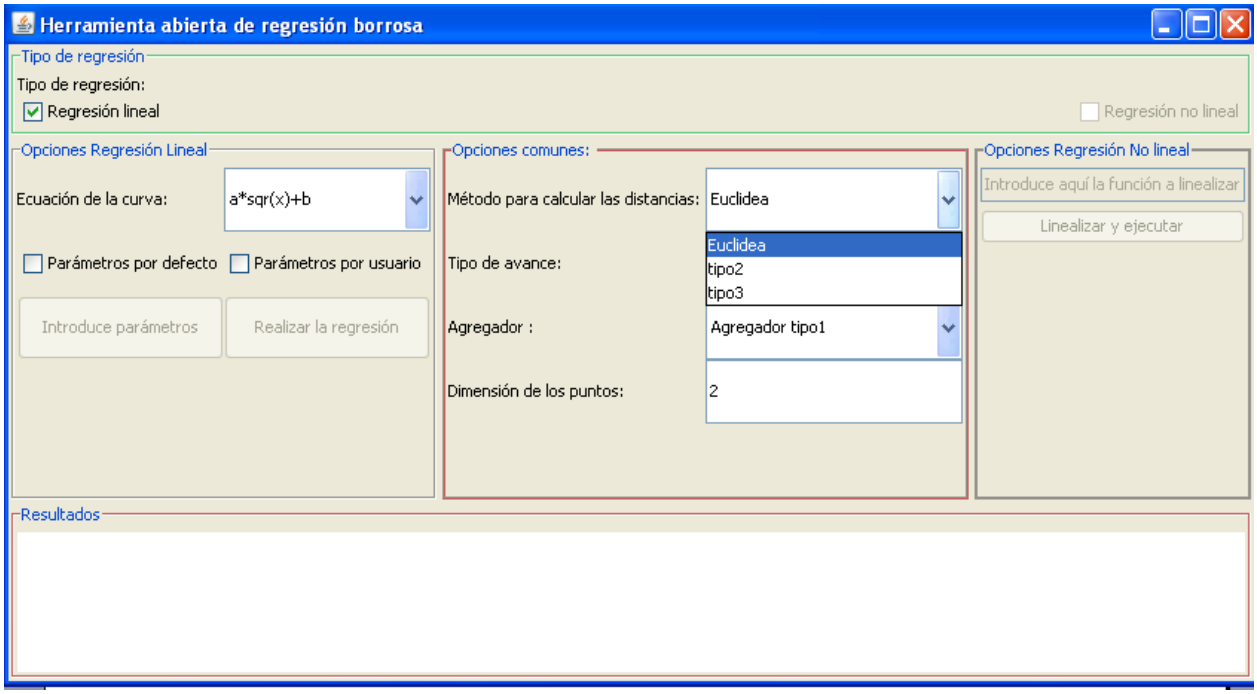


Para elegir la ecuación hacemos clic sobre el menú desplegable:

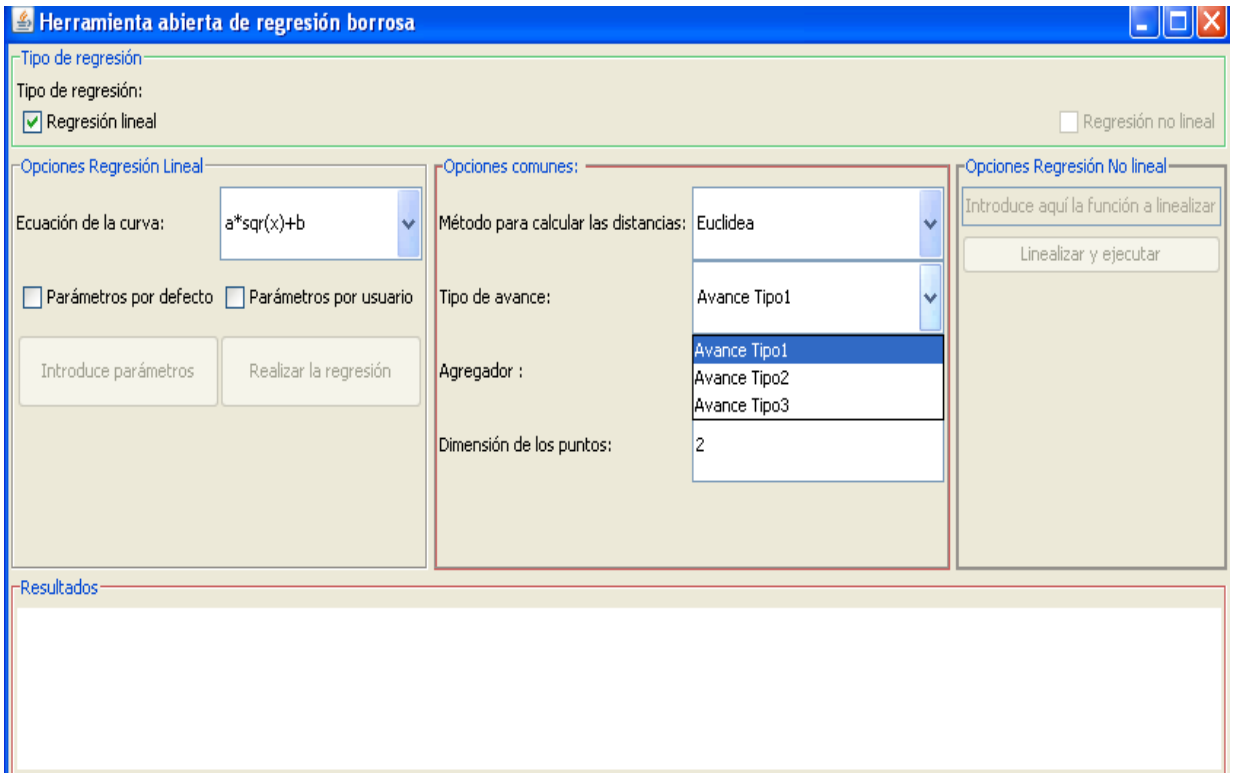
En primer lugar elegimos el tipo de ecuación para realizar la regresión, haciendo click en el menú desplegable aparecen las ecuaciones instaladas en el proyecto. Haciendo un solo click sobre la elegida ésta se queda seleccionada.

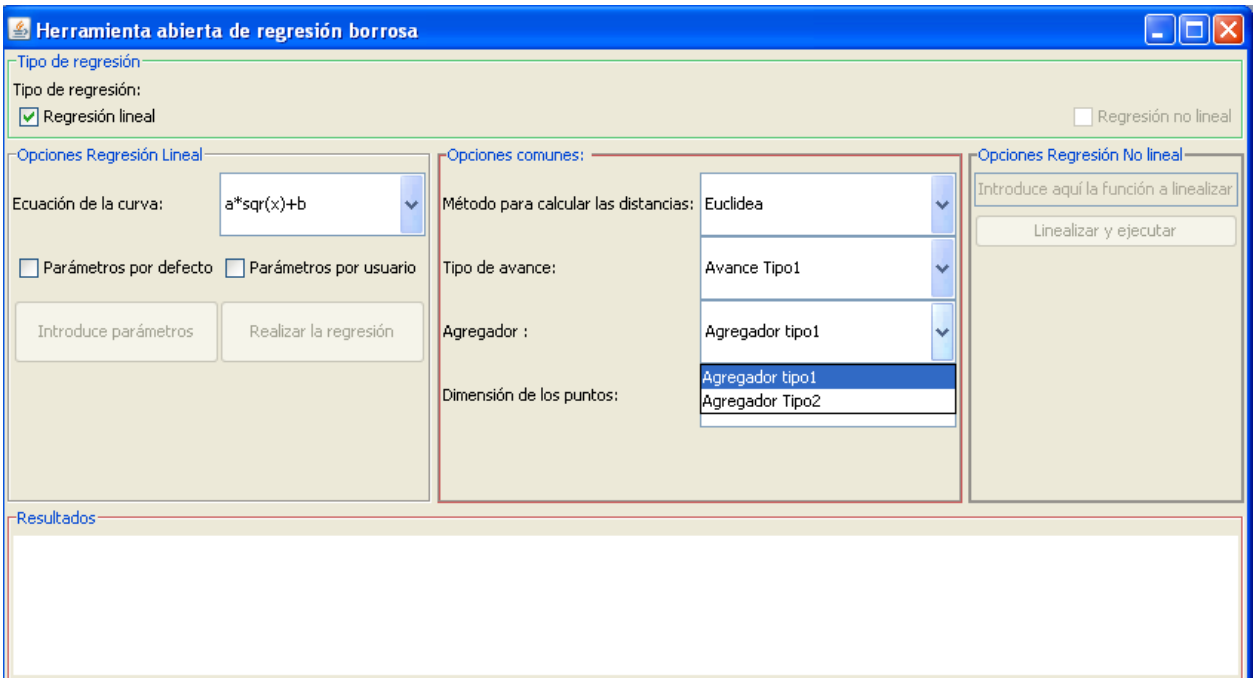


Debemos seguir el mismo proceso para elegir el método que calcula las distancias entre puntos borrosos.



Igualmente para el tipo de avance de los parámetros y el agregador para el cálculo de la distancia.

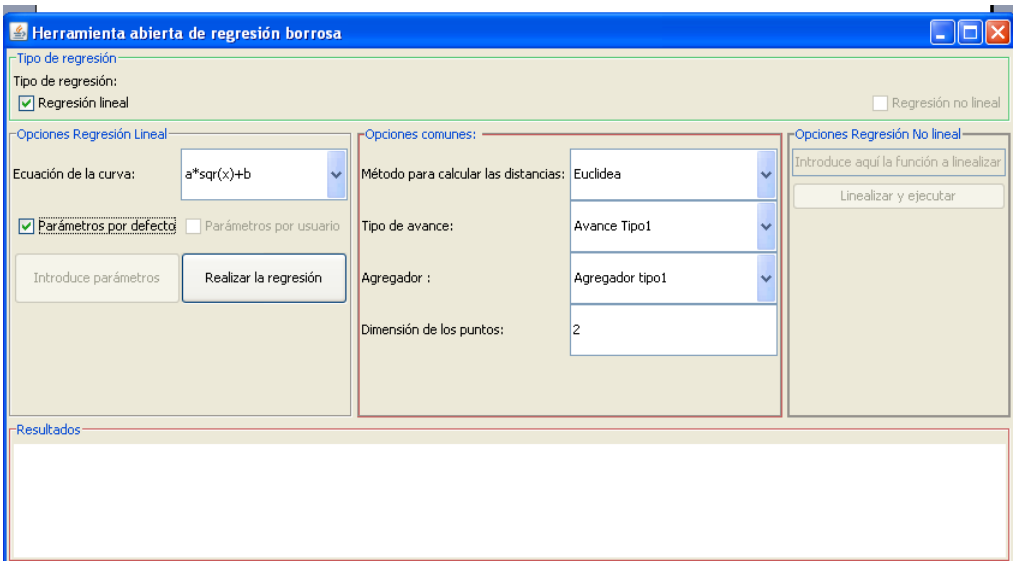




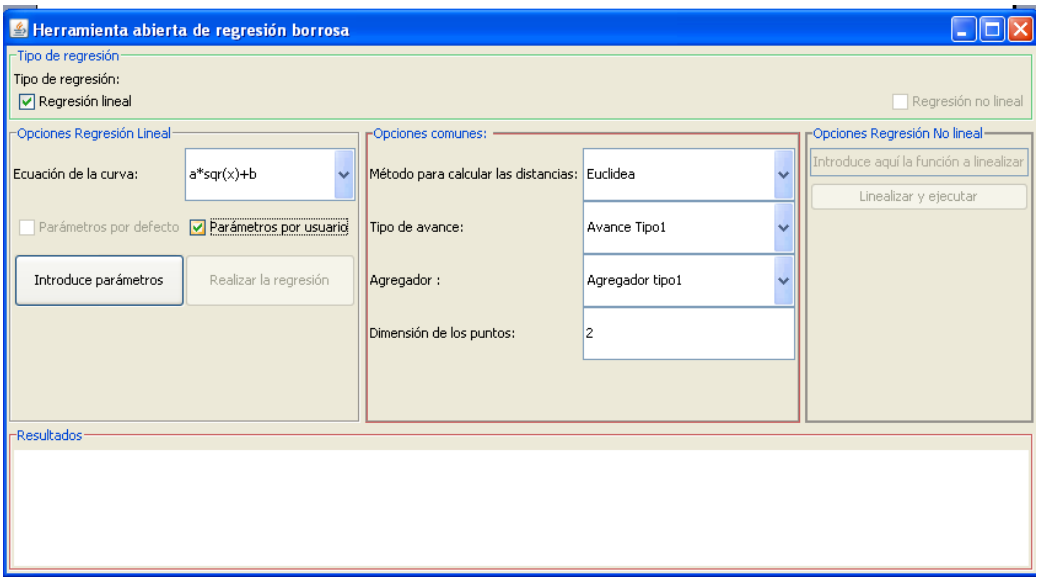
En el campo dimensión de los puntos introduciremos un entero, para fijar la dimensión de los puntos.

Para poder realizar la regresión y que el botón de ejecución se habilite debemos marcar si queremos los parámetros por defecto o introducirlos nosotros.

Si marcamos por defecto, se crean los parámetros en el motor de manera automática y se habilita el botón de regresión.

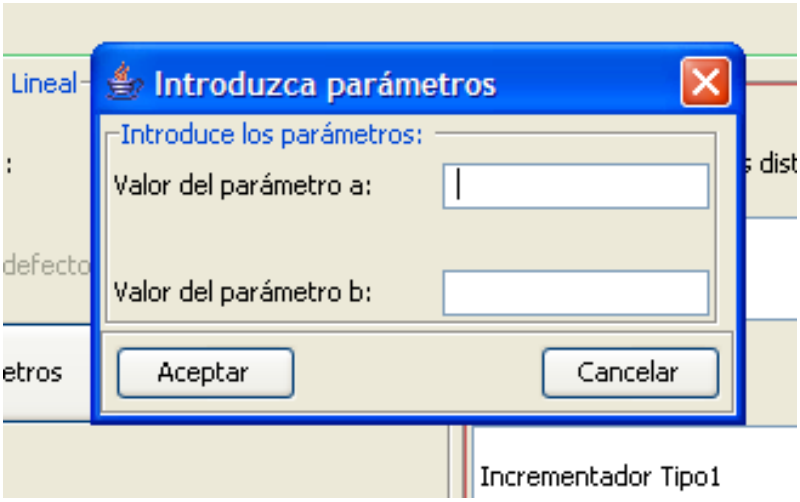


Si elegimos introducir nosotros los parámetros iniciales, se habilita el botón introduce parámetros:

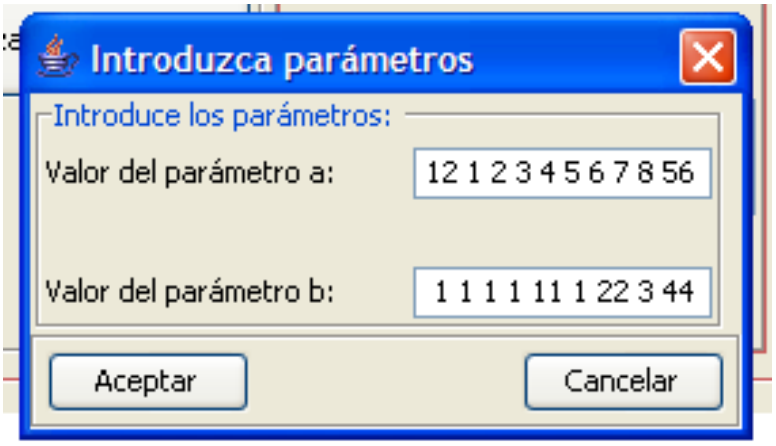


Pulsando sobre el botón de parámetros aparece la siguiente gui para introducir los valores que forman el punto borroso de cada parámetro (los parámetros son puntos borrosos).

La gui que se muestra es en el caso en que la ecuación elegida es la primera (solo tiene como parámetros a y b).

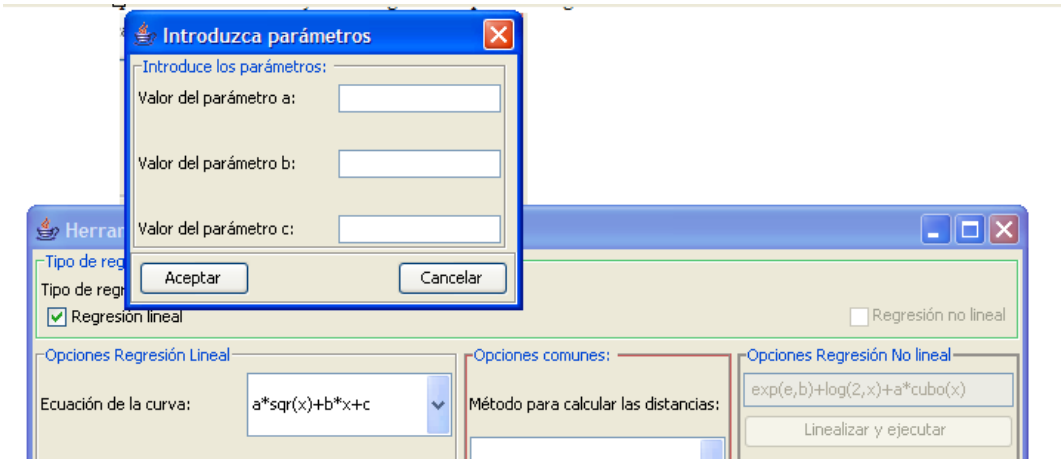


Una posible representación de los números borrosos puede ser escribir cada entero que forma el arraylist del punto borroso uno detrás de otro separados por un espacio. Si estamos utilizando dos dimensiones y cada dimensión está formada por un arraylist de 5 enteros, el número estará formado por 10 números seguidos, los 5 primeros para la primera dimensión y los 5 siguientes para la segunda.



La gui tiene un método privado que se encarga de recoger los valores del textBox e introducirlos en dos puntos borrosos que pasan a ser el parámetro a y el parámetros b.

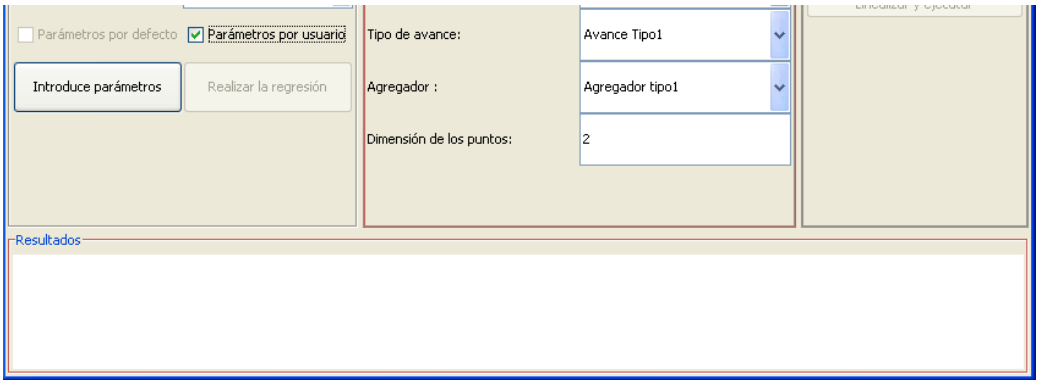
Si eligiésemos otra ecuación con más parámetros aparecería otra gui:



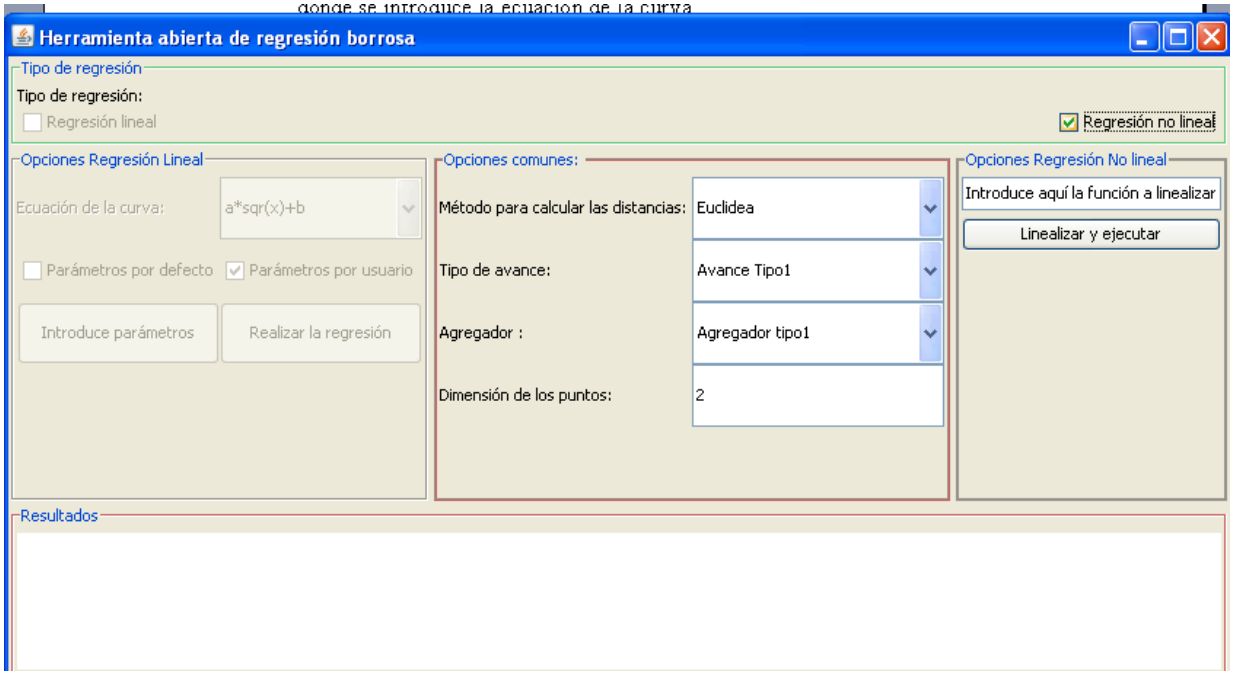
Esta idea de recogida de parámetros tiene la ventaja de ser bastante sencilla, sin embargo el usuario tiene que saber como son los puntos borrosos dentro del motor.

Por tanto, si queremos que el usuario no tenga por que saber como son los puntos podemos hacer la recogida de parámetros similar a la recogida de puntos del otro grupo en el que utilizan las funciones de pertenencia para transformar los puntos usuario a punto motor.

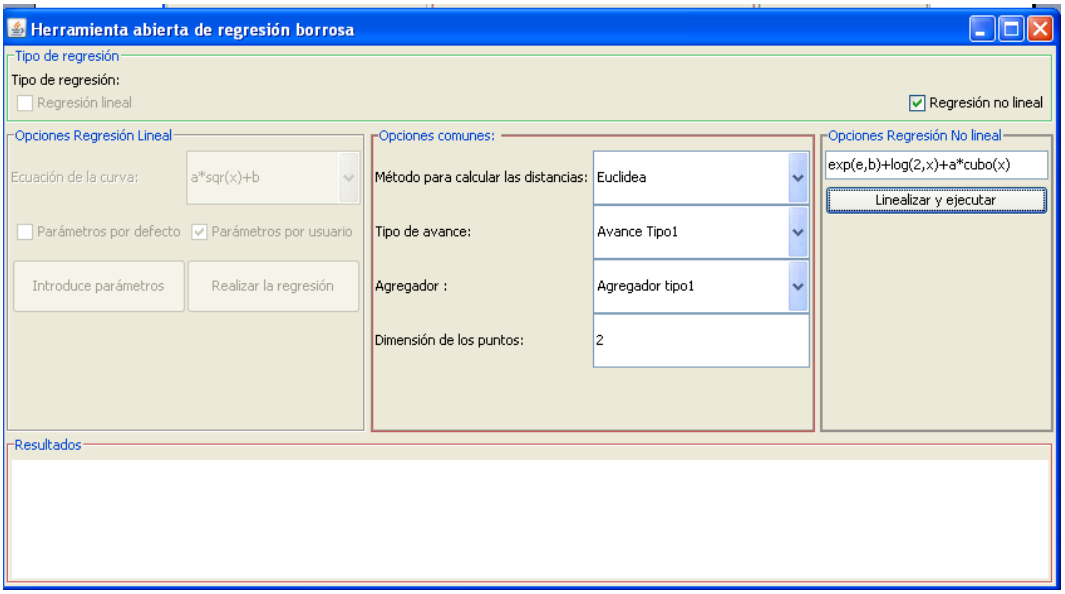
Una vez que los parámetros están creados ya podemos realizar la regresión, los valores de los parámetros de la curva para los que la regresión es óptima saldrán en el textArea de la zona de resultados:



Regresión no lineal.
Se habilitan las siguientes opciones:
El método para calcular las distancias y el avance de parámetros y un textÁrea donde se introduce la ecuación de la curva.



Debemos introducir la ecuación donde se indica:
Si queremos utilizar la ecuación: exp(e,b)+log(2,x)+a*cubo(x)
Su significado en aritmética entera sería: el número e elevado a b + el logaritmo en base 2 de x + a por x al cubo.
En el apéndice D se incluye un manual de cómo introducir ecuaciones correctamentes.



9.2. *Apendice B : Ampliación de fundamentos matemáticos*

9.2.1. Definicion de conjunto borroso

Sea U un universo de discusión con su elemento genérico denotado por u. Luego un subconjunto difuso A de U está caracterizado por una función de pertenencia:

$$\mu_A: U \longrightarrow [0,1]$$

que asocia a cada elemento u de U un número $\mu_A(u)$ que representa el grado de pertenencia de u en A. A se denota como el conjunto de pares ordenados $\{\mu_A(u), u\}$.

Es decir un conjunto difuso A se considera como un conjunto de pares ordenados, en los que el primer componente es un número en el rango [0,1] que denota el grado de pertenencia de un elemento u de U en A, y el segundo componente especifica precisamente quién es ése elemento de u.

En general los grados de pertenencia son subjetivos en el sentido de que su especificación es una cuestión objetiva. Se debe aclarar que aunque $\mu_A(u)$ puede interpretarse como el grado de verdad de que la expresión "u ∈A" sea cierta, es más natural considerarlo simplemente como un grado de pertenencia.

Puede notarse además que:

a) Mientras más próximo está $\mu_A(u)$ al valor 1, se dice que u pertenece más a A (de modo que 0 y 1 denotan la no pertenencia y la pertenencia completa, respectivamente).

b) Un conjunto en el sentido usual es también difuso pues su función característica

$$\mu_A : u \rightarrow \begin{cases} 0 \text{ si } \mu \in A \\ 1 \text{ si } \mu \notin A \end{cases}$$

es también una función μ_A:u⟶ [0,1]; o sea que los conjuntos difusos son una generalización de los conjuntos usuales [ACIS3].

Las operaciones básicas de los conjuntos difusos son:

- Contención o subconjunto:

  A es un subconjunto de B si y solo si $\mu_A(x) \leq \mu_B(x)$, para todo x
  $A \subseteq B \Leftrightarrow \mu_A(x) \leq \mu_B(x)$

- Unión:

  La unión de los conjuntos difusos A y B es el conjunto difuso C, y se escribe como C= AOR B, su función de dependencia está dada por
  $\mu_C(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) - \mu_B(x)$

- Intersección:

  La intersección de los conjuntos difusos A y B es el conjunto difuso C, y se escribe como C=A AND B, su función de dependencia está dada por $\mu_C(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \cdot \mu_B(x)$

- Complemento (negación):

  El complemento del conjunto difuso A, denotado por Ā (¬A, NOT A), se define como
  $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$

- Producto Cartesiano:

  Si A y B son conjuntos difusos en X e Y, el producto cartesiano de los conjuntos A y B A x B en el espacio de X x Y tiene la función de pertenencia
  $\mu_{AxB}(x,y) = \min(\mu_A(x), \mu_B(x))$

- Co-producto Cartesiano

  A + B en el espacio X x Y tiene la función de pertenencia
  $\mu_{AxB}(x,y) = \max(\mu_A(x), \mu_B(x))$ [aaa]

La modelización de la información aportada por el ser humano precisa de modelos que estimen estas diferentes formas de imprecisión. En esta herramienta se propone para su modelización el uso de la Teoría de Conjuntos Borrosos y marcos matemáticos relacionados.

En el modelo de regresión existen diversos valores en los que puede haber imprecisión. Estos son las entradas, las salidas, los parámetros de la curva, la distancia entre los puntos y la curva y la ponderación.

Números borrosos

Para dar borrosidad a un valor nítido, se le asignará una función de pertenencia μ:

$$\mu_{x_\beta}(x) \in [0,1]$$

El único requisito que debe cumplir el grado de pertenencia μ es que su rango debe ser el intervalo [0,1]
En adelante, la borrosidad se notará con un subíndice β.

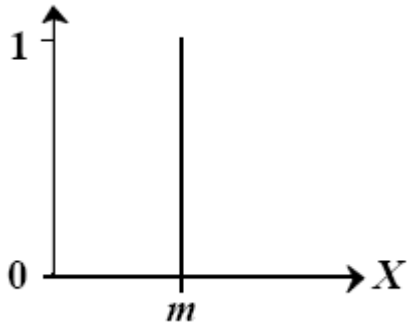
Distintas funciones para expresar el grado de pertenencia

Algunas de las funciones de pertenencia encontradas más frecuentemente en la literatura son [Gal05], [Car01]:

- Singleton:

  Está definida por un valor m, para el cual el valor de la función es uno, siendo 0 para los demás:

$$\mu(x) = \begin{cases} 1 & \text{si } x = m \\ 0 & \text{si } x \neq m \end{cases}$$



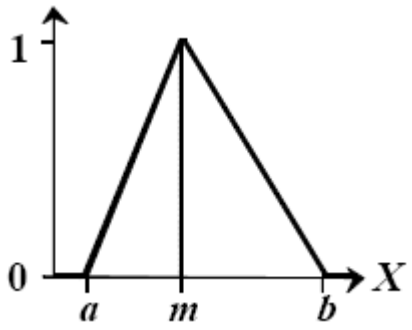
  Un número nítido puede definirse como un número borroso con una función de pertenencia singleton asociada, cuyo m es el valor del número nítido [ACIS03]:

$$a = 7 \qquad \mu_a(x) = \begin{cases} 1 & \text{si } x = 7 \\ 0 & \text{si } x \neq 7 \end{cases}$$ a.

- Triangular:

  Definida por sus límites superior e inferior a y b y un valor modal m tal que a<m<b

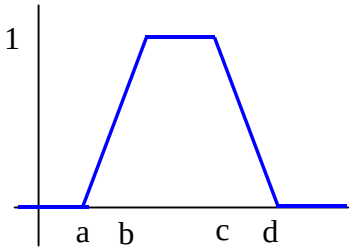
$$\mu(x)=\begin{cases}0 & \text{si } x\le a \text{ o } x\ge b\\ \dfrac{x-a}{m-a} & \text{si } x\in(a,m]\\ \dfrac{b-x}{b-m} & \text{si } x\in(m,b)\end{cases}$$



- Trapezoidal:

  Definida por cuatro parámetros a, b, c y d tales que a < b < c < d

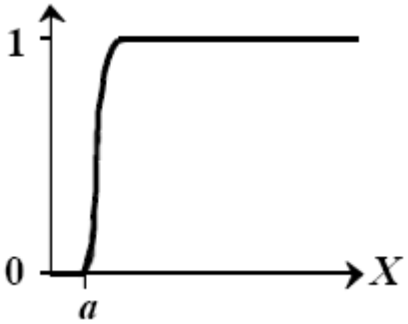
$$\mu(x)=\begin{cases}0 & \text{para } x\le a\\ \dfrac{x-a}{b-a} & \text{para } a<x\le b\\ 1 & \text{para } b<x\le c\\ \dfrac{d-x}{b-c} & \text{para } c<x\le d\\ 0 & \text{para } x>d\end{cases}$$



- Gamma:

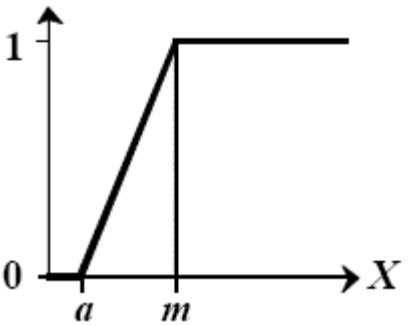
  Esta función queda definida por su límite inferior a y un valor k > 0. A mayor k más rápido es el crecimiento de la función.

$$\mu(x)=\begin{cases}0 & \text{si } x\le a\\ 1-e^{-k(x-a)^2} & \text{si } x>a\end{cases}$$



  Se puede aproximar linealmente mediante la siguiente función:

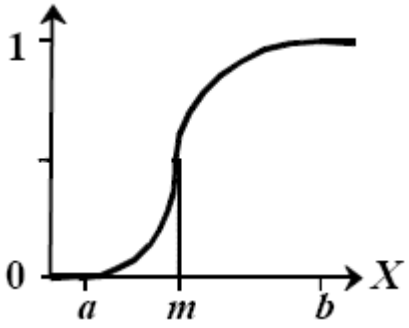
$$\mu(x)=\begin{cases}0 & \text{si } x\le a\\ \dfrac{x-a}{m-a} & \text{si } a<x<m\\ 1 & \text{si } x\ge m\end{cases}$$



- Función S:

  Definida por sus límites superior e inferior a y b y un punto de inflexión m tal que a<m<b

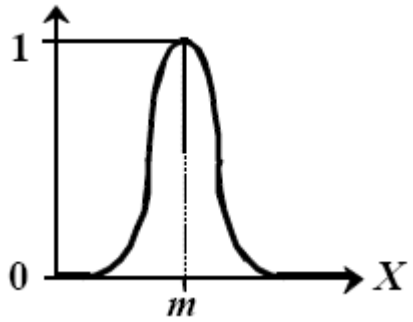
$$\mu(x)=\begin{cases}0 & \text{si } x\le a \text{ o } x\ge b\\ 2\cdot\left(\dfrac{x-a}{b-a}\right)^2 & \text{si } x\in(a,m]\\ 1-2\cdot\left(\dfrac{b-x}{b-m}\right)^2 & \text{si } x\in(m,b)\end{cases}$$



- Gaussiana:

  Definida por su valor medio m y un valor k > 0. Tiene forma de campana, siendo más estrecha cuanto mayor es el valor k.

$$\mu(x)=e^{-k(x-m)^2}$$



Puntos borrosos

Un punto se considerará borroso si alguna de sus componentes es borrosa. En ese caso, el punto se notará como $p_{\beta i}$:

$$p_{\beta i}=\left(y_{\beta i},\vec{x}_{\beta i}\right)$$

siendo

$$\vec{x}_{\beta i}=\left(x_{\beta i1},\ldots,x_{\beta in}\right)$$

y las funciones de pertenencia asociadas

$$\mu_{x_{\beta ij}}(x)$$

$$j=1\ldots n$$

y

$$\mu_{y_{\beta i}}(x)$$

En el caso de que la componente sea nítida, la función de pertenencia asociada será una singleton con un valor para su parámetro m igual al valor de la componente (véase a.).

A efectos de conseguir mayor claridad en las fórmulas, dado un punto $p_{\beta i}$, la componente $y_{\beta i}$ se notará como $p_{\beta i0}$, y cada $x_{\beta ij}$ como $p_{\beta ij}$, por lo que un punto puede expresarse como:

$$p_{\beta i} = (y_{\beta i}, x_{\beta i1}, \ldots, x_{\beta in}) = (p_{\beta i0}, p_{\beta i1}, \ldots, p_{\beta in}) \qquad \text{b.}$$

La función de pertenencia de un punto borroso $P_{\beta i}$ se puede considerar como una agregación de las funciones de pertenencia de cada uno de los números borrosos que lo componen. Así pues, cada punto borroso tendrá asociado un agregador (Ω_{pi}) para obtener el valor de su función de pertenencia. Esta se calculará de la siguiente manera:

$$\mu_{P_{\beta i}}(p) = \Omega_{Pi}^{n}{}_{j=0} \left(\mu_{p_{\beta ij}}(p_j) \right)$$

siendo p un punto nítido de la forma:

$$p = (y, x_1, \ldots, x_n) = (p_0, p_1, \ldots, p_n)$$

y $p_{\beta i}$ un punto borroso

[BCF05]

9.2.2. Concepto de regresión

El Análisis de Regresión es una técnica estadística que tiene como objetivo establecer modelos matemáticos para representar formalmente las relaciones de dependencia existente entre un conjunto de variables estadísticas.
Tanto en el caso de dos variables (regresión simple) como en el de más de dos variables (regresión múltiple), el análisis de regresión lineal puede utilizarse para explorar y cuantificar la relación entre una variable llamada dependiente o criterio (Y) y una o más variables llamadas independientes o predictoras (X1, X2, ..., Xk), así como para desarrollar una ecuación lineal con fines predictivos. Además, el análisis de regresión lleva asociados una serie de procedimientos de diagnóstico (análisis de los residuos, puntos de influencia) que informan sobre la estabilidad e idoneidad del análisis y que proporcionan pistas sobre cómo perfeccionarlo.
Se adapta a una amplia variedad de situaciones. En la investigación social, el análisis de regresión se utiliza para predecir un amplio rango de fenómenos, desde medidas económicas hasta diferentes aspectos del comportamiento humano. En el contexto de la investigación de

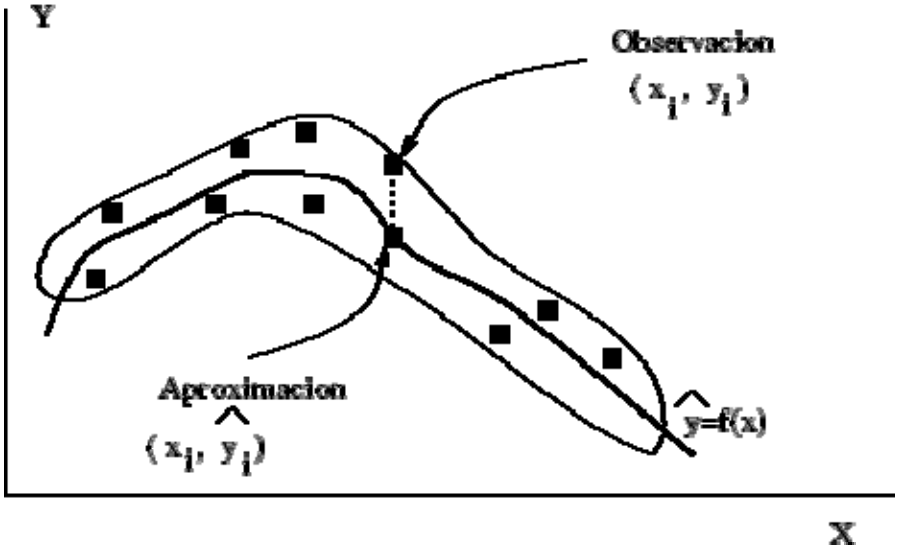
mercados puede utilizarse para determinar en cuál de diferentes medios de comunicación puede resultar más eficaz invertir; o para predecir el número de ventas de un determinado producto. En física se utiliza para caracterizar la relación entre variables o para calibrar medidas, etc.

Sean {(xi,yj); nij i=1,...,r; j=1,...,c} los valores y la distribución de frecuencias conjunta de las variables analizadas X e Y. Se denomina diagrama de dispersión o nube de puntos a la representación, en un sistema de ejes cartesianos (X, Y), de los valores observados de las variables, en el que a cada par (xi,yj) se le asocia su frecuencia conjunta de observación nij.

Esta representación es conveniente como paso previo al análisis de regresión en tanto que la forma de la nube de puntos permite obtener una idea inicial del tipo de dependencia existente entre X e Y.

Un diagrama de dispersión ofrece una idea bastante aproximada sobre el tipo de relación existente entre dos variables. Pero, además, un diagrama de dispersión también puede utilizarse como una forma de cuantificar el grado de relación lineal existente entre dos variables: basta con observar el grado en el que la nube de puntos se ajusta a una línea recta.

Mediante las técnicas de regresión de una variable Y sobre una variable X, se busca una función que sea una buena aproximación de una nube de puntos (x_i,y_i), mediante una curva del tipo $\hat{Y} = f(X)$ [BMA04]. Para ello la diferencia entre los valores y_i e $\hat{y}_i$ ha de ser tan pequeña como sea posible.



En el cálculo de los coeficientes del modelo de regresión lineal simple intervienen, aparte de la media, la varianza y la covarianza .

.La varianza de Y se expresa mediante : $s_Y^2 = \frac{\sum_{i=1}^{n} (y_i - \overline{y})^2}{n}$.

La covarianza de n valores $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ de (X,Y) indica si la posible relación entre dos variables es directa o inversa.

$$S_{xy} = \frac{1}{n}\sum_i (x_i - \bar{x})(y_i - \bar{y})$$

- Directa: $S_{xy} > 0$. La pendiente de la recta también es positiva.
- Inversa: $S_{xy} < 0$. La pendiente de la recta también es negativa.
- Incorreladas: $S_{xy} = 0$. En este caso las rectas de regresión serán paralelas a los ejes de ordenadas.

  El signo de la covarianza informa si el aspecto de la nube de puntos es creciente o no, pero no dice nada sobre el grado de relación entre las variables.
  [BCF05]

9.3. Apéndice C: Bibliografía

[Gal05] José Galindo, “Curso Introductorio de Conjutos y Sistemas Difusos”, extraido en Junio de 2007 http://www.lcc.uma.es/~ppgg/FSS/

[BCF05] Raquel Ballester Lorenzo, Jose Ignacio del Campo Montejo, Gonzalo Flórez Puga “FORT: Una herramienta de Regresión Borrosa” 2005

Julio H Cole “Nociones de Regresión Lineal” extraido en Junio de 2007 http://www.fce.ufm.edu/Catedraticos/jhcole/Nociones.doc

[ChA01] Y.O. Chang and B.M. Ayyub. “Fuzzy regression methodos – a comparative asseessment. Fuzzy Sets and Systems”, 2001.

[JCr02] Javier Crespo, “Modelo Paramétrico Matemático Difuso para la estimación de Esfuerzo de Desarrollo del Software”, tesis doctoral, 2002.

[Bla05] Paul E. Black, ed., NIST, "Dictionary of Algorithms and Data Structures”, Enero, 2005
http://www.nist.gov/dads/HTML/manhattanDistance.html
http://www.nist.gov/dads/HTML/lmdistance.html

[FGL05] UNIPHIZ Lab software UNIPHIZ Lab, 2005 “Find Graph Quick and Easy Graphing, digitizing and curve fitting software”.Extraído en Junio de 2005 http://www.uniphiz.com/findgraph.htm

[RLM00] Técnicas de regresión: Regresión Lineal Múltiple. Pértega Díaz, S., Pita Fernández, S.,Unidad de Epidemiología Clínica y Bioestadística. Complexo Hospitalario Juan Canalejo. A Coruña (España). CAD ATEN PRIMARIA 2000; 7: 173-176.