**Trabajo Práctico**

**Matemática Superior**

1° Cuatrimestre

2015

**Profesora: María Alicia Piñeiro**

**Curso:K3051**

**Alumnos:**

|  |  |
| --- | --- |
| **Nombre** | **Legajo** |
| Cardoso, Ariel | 146.525-9 |
| Fernández, Lautaro | 146.524-7 |

**Consigna:**

a) Grafique la nube de puntos de los 10 datos. Y calcule de cuantas formas diferentes podrían corresponder las 8 últimas muestras, sabiendo que son 4 de cada túnel.

b) Dado que los valores de las muestras pueden tener pequeños errores de mediciones o estimaciones de las cantidades, indique por que el método mas conveniente para hallar la expresión lineal que caracteriza a cada grupo de muestras es el de Mínimos Cuadrados.

c) Determine una estrategia para determinar cual de todas las posibilidades es la mejor. (Puede hacer un programa que mediante un algoritmo vaya calculando una por una y comparando, puede hacerlo en una planilla de cálculo, o puede hacerlo a mano, la elección es a su criterio). La opción correcta es la que tenga error cuadrático (suma de los cuadrados de distancias) inferior a 0,3 en ambas rectas.

d) En base a lo realizado en el ítem c) indique el grupo de muestras de cada túnel junto a cada recta que los ajusta.

**Desarrollo:**

a) Gráfico de la nube de puntos:

VER GRÁFICO EN EXCEL

Posibilidades diferentes:

b)

Creemos conveniente la utilización del método de mínimos cuadrados y aproximación lineal debido al comportamiento de las muestras y a que sabemos que si interpoláramos, el tiempo empleado y el resultado no serían representativos del problema en cuestión.

c)

Para encontrar la mejor posibilidad de combinaciones de los elementos, decidimos desarrollar una aplicación de formularios de Windows en el lenguaje C#, que permita el ingreso de un número par de muestras compuestas de 2 coordenadas cada una, y encuentre la mejor combinación de ellas en 2 conjuntos, siendo la mejor combinación la que:

* El error sea la mayor sumatoria de distancias al cuadrado de cada una de las rectas.
* Éste error sea menor a la sumatoria de distancias al cuadrado de cada una de las rectas de cada una de las otras combinaciones.

De ésta manera, la aplicación almacena las posibles combinaciones de N en N/2, y prueba de a 2 conjuntos de N/2 elementos de todas las posibles, siendo un par de conjuntos válido aquel cuyo primer conjunto de elementos no tenga ningún elemento del segundo conjunto.

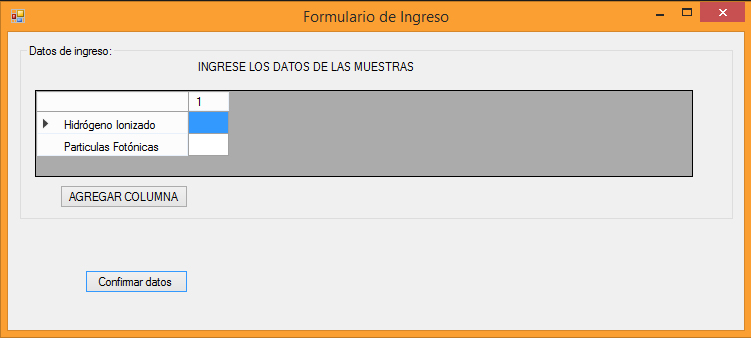
Luego, la aplicación prueba los conjuntos válidos obtenidos de la siguiente manera:

Por cada conjunto de muestras del par válido:

* Realiza la suma de las X.
* Realiza la suma de las X2.
* Realiza la suma de las Y.
* Realiza la suma de las Y\*X.
* Resuelve el sistema dado y almacena la recta solución en una lista de rectas denominada RectasResultantes.
* Realiza la diferencia de distancias cuadradas entre los elementos y la recta solución.
* Compara la diferencia con la menor hallada:
  + Si es menor, continúa hasta que termine de probar el par.
  + Si es mayor, descarta el par.
* Al terminar de probar el par y no descartarlo, se reemplaza la menor diferencia hallada por la mayor diferencia de ambas rectas y se guarda el par por ser el actual mejor.

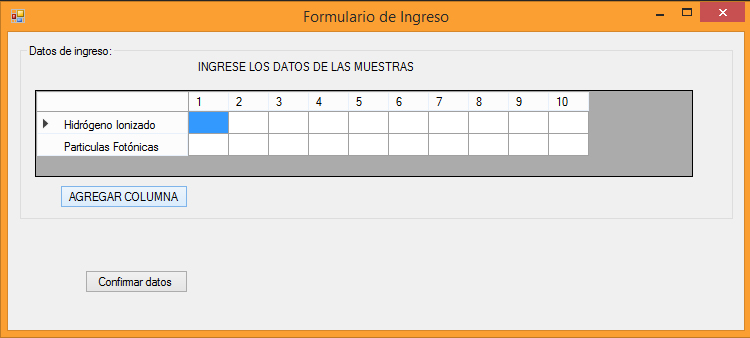
Cuando la aplicación termine de probar todos los pares de conjuntos de muestras posibles, se queda con el mejor par encontrado y lo informa al usuario.

1. Formulario principal vacío:



En este formulario se ingresa la cantidad de muestras que el usuario desee, pudiendo agregar nuevas con el botón “AGREGAR COLUMNA”.

2.

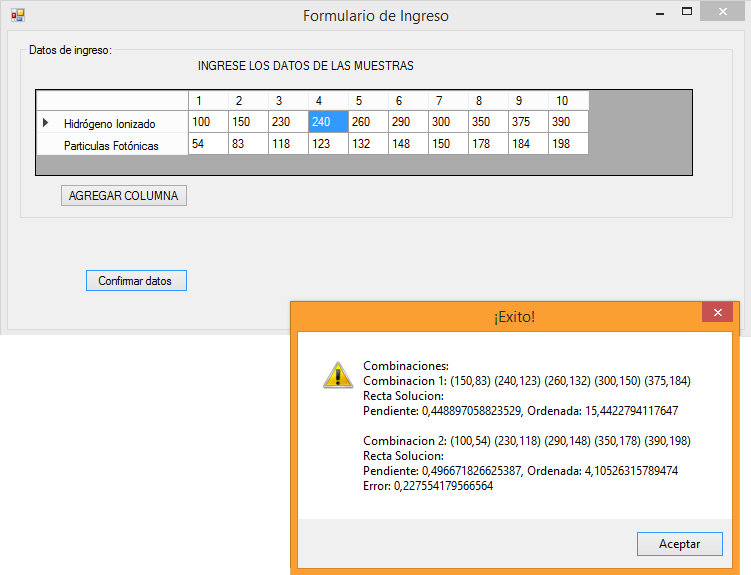


Aquí se muestra la tabla agregando 10 columnas.

3.

Aquí se muestra la tabla completa con los valores del enunciado.

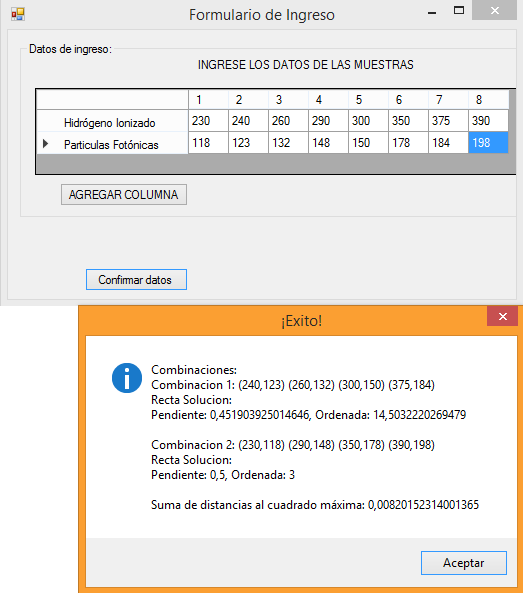
4.



Este es el resultado que arroja el programa al confirmar los datos:

El par de conjuntos de elementos, cada uno con los elementos que lo componen y su recta solución con pendiente y ordenada al origen.

Además se informa la mayor sumatoria al cuadrado obtenida, que al ser menor a 0,3, resulta ser correcta.

5.

También podemos probar con los 8 puntos restantes y obtenemos una diferencia de distancias al cuadrado menor a 0,01, y luego las clasificamos con las muestras que conocemos.

**Anexo Código Fuente:**

public partial class Form1 : Form

{

List<Muestra> muestras = new List<Muestra>();

int tamanioMuestra = 10;

public Form1()

{

InitializeComponent();

DataGridViewRow \_fila=new DataGridViewRow();

\_fila.HeaderCell.Value="Hidrógeno Ionizado";

\_dgvDatos.Rows.Add(\_fila);

DataGridViewRow \_fila2 = new DataGridViewRow();

\_fila2.HeaderCell.Value = "Particulas Fotónicas";

\_dgvDatos.Rows.Add(\_fila2);

\_dgvDatos.RowHeadersWidthSizeMode=

DataGridViewRowHeadersWidthSizeMode.AutoSizeToAllHeaders;

}

private bool esPar(int numero)

{

return numero % 2 == 0;

}

private void b\_ingresarValor\_Click\_1(object sender, EventArgs e)

{

if (esPar(\_dgvDatos.Columns.Count))

{

List<Muestra> \_listaDeMuestras = new List<Muestra>();

for (int i = 0; i < \_dgvDatos.Columns.Count; i++)

{

//agregar muestras a una lista de muestras

Muestra \_nuevaMuestra = new Muestra(

Convert.ToDouble(\_dgvDatos.Rows[0].Cells[i].Value),

Convert.ToDouble(\_dgvDatos.Rows[1].Cells[i].Value));

\_listaDeMuestras.Add(\_nuevaMuestra);

}

Combinador \_combinador = new Combinador();

int n = \_listaDeMuestras.Count;

int p = Convert.ToInt32(Math.Truncate(Convert.ToDouble(\_listaDeMuestras.Count/2)));

//Armo una lista de listas de muestras con las combinaciones de la lista de muestras

List<List<Muestra>> \_listaDeListaDeMuestras = new List<List<Muestra>>();

\_listaDeListaDeMuestras = \_combinador.realizarCombinaciones(\_listaDeMuestras, p);

//Creo error, primero

double error = 0;

bool primero = true;

//Solucion parcial va a ser la mejor solución posible con el menor error

List<List<Muestra>> solucionParcial = new List<List<Muestra>>();

//Para cada lista de muestras en la lista de lista de muestras

\_listaDeListaDeMuestras.ForEach(delegate(List<Muestra> unaListaDeMuestras)

{

//para cada otra lista de muestras en la lista de lista de muestras

\_listaDeListaDeMuestras.ForEach(delegate(List<Muestra> otraListaDeMuestras)

{

//Si tienen todos los elementos diferentes

if (unaListaDeMuestras.TrueForAll(elemento =>

!otraListaDeMuestras.Exists(otroElemento => elemento == otroElemento)))

{

//Los agrego a otra lista de lista de muestras

List<List<Muestra>> listaDeListaDeMuestras = new List<List<Muestra>>();

listaDeListaDeMuestras.Add(unaListaDeMuestras);

listaDeListaDeMuestras.Add(otraListaDeMuestras);

//Trato de resolver el sistema

Combinacion \_combinacion = new Combinacion(listaDeListaDeMuestras);

Resultado \_resultado = \_combinacion.calcularRectas();

//Si es la primera combinación probada

if(primero==true)

{

//Para cada estructura de parametros de muestras

\_resultado.\_listaParametrosDeMuestras.ForEach(delegate(ParametrosListaDeMuestras unosParametros)

{

//Si es la primer estructura de parámentros de muestra

if (primero == true)

{

//El error es igual al error del primero

error= unosParametros.\_sumaDeDistanciasAlCuadrado;

primero = false;

//Es la nueva solución parcial

solucionParcial.Add(unaListaDeMuestras);

solucionParcial.Add(otraListaDeMuestras);

}

else

{

//Si el error es mayor, lo asigno a error

if(unosParametros.\_sumaDeDistanciasAlCuadrado>error)

{

error = unosParametros.\_sumaDeDistanciasAlCuadrado;

}

}

});

}

//Si el error del resultado es menor al error actual

if (\_resultado.errorMenorALoIndicado(error))

{

//Para cada estructura de parámetros del resultado

error=\_resultado.\_listaParametrosDeMuestras.Max

(elemento => elemento.\_sumaDeDistanciasAlCuadrado);

//Ésta pasa a ser la solución parcial

solucionParcial.Clear();

solucionParcial.Add(unaListaDeMuestras);

solucionParcial.Add(otraListaDeMuestras);

}

}

});

}

);

//Muestro la solución definitiva

List<Muestra> primerLista = solucionParcial.ElementAt(0);

List<Muestra> segundaLista = solucionParcial.ElementAt(1);

Combinacion combinacionDefinitiva = new Combinacion(solucionParcial);

Resultado resultadoDefinitivo = combinacionDefinitiva.calcularRectas();

string cadenaExito = "Combinaciones: ";

cadenaExito += "\nCombinacion 1: ";

primerLista.ForEach(elemento => cadenaExito =

cadenaExito + string.Format("({0},{1}) ",

elemento.ParticulasFotonicas, elemento.HidrogenoIonizado));

cadenaExito += string.Format("\nRecta Solucion:\nPendiente: {0}, Ordenada: {1}\n"

, combinacionDefinitiva.\_rectasSolucion[0].Pendiente, combinacionDefinitiva.\_rectasSolucion[0].Ordenada);

cadenaExito += "\nCombinacion 2: ";

segundaLista.ForEach(elemento => cadenaExito =

cadenaExito + string.Format("({0},{1}) ",

elemento.ParticulasFotonicas, elemento.HidrogenoIonizado));

cadenaExito += string.Format("\nRecta Solucion:\nPendiente: {0}, Ordenada: {1}"

, combinacionDefinitiva.\_rectasSolucion[1].Pendiente, combinacionDefinitiva.\_rectasSolucion[1].Ordenada);

cadenaExito += string.Format("\n \nSuma de distancias al cuadrado máxima: {0}", error);

MessageBox.Show(cadenaExito,

"¡Exito!", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show(string.Format("Debe ingresar una cantidad par de muestras."),

"¡Error!", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void button1\_Click(object sender, EventArgs e)

{

int cantidadFilas=\_dgvDatos.Columns.Count+1;

DataGridViewColumn \_columnaNueva =

new DataGridViewColumn(\_dgvDatos.Columns[0].CellTemplate);

\_columnaNueva.Width = 40;

\_columnaNueva.Name="Columna"+cantidadFilas.ToString();

\_columnaNueva.HeaderText = cantidadFilas.ToString();

\_dgvDatos.Columns.Add(\_columnaNueva);

}

private void \_dgvDatos\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

}

public class Muestra

{

private double \_particulasFotonicas, \_hidrogenoIonizado;

public double ParticulasFotonicas

{

get { return \_particulasFotonicas; }

set { \_particulasFotonicas = value; }

}

public double HidrogenoIonizado

{

get { return \_hidrogenoIonizado;}

set { \_hidrogenoIonizado = value; }

}

public Muestra(double particulasFotonicas, double hidrogenoIonizado)

{

ParticulasFotonicas = particulasFotonicas;

HidrogenoIonizado = hidrogenoIonizado;

}

public bool esIgual(Muestra otraMuestra)

{

return ((ParticulasFotonicas == otraMuestra.ParticulasFotonicas) && (HidrogenoIonizado == otraMuestra.HidrogenoIonizado));

}

public double distanciaAlcuadradoConRespectoA(Recta recta)

{

double valorDeYEnLARecta = recta.YPara(ParticulasFotonicas);

double distancia = valorDeYEnLARecta - HidrogenoIonizado;

return distancia\*distancia;

}

}

public class Combinador

{

List<List<Muestra>> \_combinacionesPosibles=new List<List<Muestra>>();

List<Muestra> \_combinacionActual=new List<Muestra>();

int cantidadElementos;

public List<List<Muestra>> realizarCombinaciones(List<Muestra> \_muestras, int p)

{

cantidadElementos = p;

combinar(\_muestras, p);

return \_combinacionesPosibles;

}

public void combinar(List<Muestra> \_muestras, int p)

{

for (int i = 0; i < \_muestras.Count &&p>0; i++)

{

\_combinacionActual.Add(\_muestras.ElementAt(i));

List<Muestra> nuevaLista = new List<Muestra>();

\_muestras.ForEach(x=>nuevaLista.Add(x));

nuevaLista.RemoveRange(0,i+1);

combinar(nuevaLista, p - 1);

List<Muestra> otraLista = new List<Muestra>();

\_combinacionActual.ForEach(x => otraLista.Add(x));

if(otraLista.Count==cantidadElementos)\_combinacionesPosibles.Add(otraLista);

\_combinacionActual.RemoveAt(\_combinacionActual.Count - 1);

}

}

}

public struct ParametrosListaDeMuestras

{

public double \_sumaX, \_sumaY, \_sumaXalCuadrado, \_sumaXporY, \_sumaDeDistanciasAlCuadrado;

//Constructor

public ParametrosListaDeMuestras(double \_sumaX, double \_sumaY,

double \_sumaXalCuadrado, double \_sumaXporY, double \_sumaDeDistanciasAlCuadrado)

{

this.\_sumaX = \_sumaX;

this.\_sumaY = \_sumaY;

this.\_sumaXalCuadrado = \_sumaXalCuadrado;

this.\_sumaXporY = \_sumaXporY;

this.\_sumaDeDistanciasAlCuadrado = \_sumaDeDistanciasAlCuadrado;

}

};

public class Combinacion

{

private List<List<Muestra>> \_muestras;

public List<Recta> \_rectasSolucion;

public List<List<Muestra>> Muestras

{

get { return \_muestras; }

set { \_muestras = value; }

}

public Combinacion(List<List<Muestra>> \_muestras)

{

Muestras = \_muestras;

}

private double sumaParticulasFotonicas(List<Muestra> muestras)

{

double sum=0;

muestras.ForEach(elemento => sum = sum + elemento.ParticulasFotonicas);

return sum;

}

private double sumaHidrogenoIonizado(List<Muestra> muestras)

{

double sum = 0;

muestras.ForEach(elemento => sum = sum + elemento.HidrogenoIonizado);

return sum;

}

private double sumaParticulasFotonicasAlCuadrado(List<Muestra> muestras)

{

double sum = 0;

muestras.ForEach(elemento => sum = sum + elemento.ParticulasFotonicas\*elemento.ParticulasFotonicas);

return sum;

}

private double sumaParticulasFotonicasPorHidrogenoIonizado(List<Muestra> muestras)

{

double sum = 0;

foreach (Muestra item in muestras)

{

sum = sum + item.HidrogenoIonizado \* item.ParticulasFotonicas;

}

return sum;

}

private Recta resolverSistema(double a, double b, double c, double d, double e, double f)

{

double pendiente=0, ordenada=0;

double det = a \* e - b \* d;

if (det != 0)

{

ordenada = (e \* c - b \* f) / det;

pendiente = (a \* f - d \* c) / det;

}

else

{

//El sistema no tiene soluciones o infinitas, pero le asignamos (0,0). Igualmente nunca sucede en este TP.

}

return (new Recta(pendiente, ordenada));

}

private double sumaDeDistanciasAlCuadrado(List<Muestra> muestras, Recta recta)

{

double sum = 0;

muestras.ForEach(elemento => sum = sum + elemento.distanciaAlcuadradoConRespectoA(recta));

return sum;

}

public ParametrosListaDeMuestras cargarParametrosDeListaDeMuestras(List<Muestra> \_listaDeMuestras)

{

double sumaX=sumaParticulasFotonicas(\_listaDeMuestras);

double sumaY=sumaHidrogenoIonizado(\_listaDeMuestras);

double sumaXalCuadrado = sumaParticulasFotonicasAlCuadrado(\_listaDeMuestras);

double sumaXporY = sumaParticulasFotonicasPorHidrogenoIonizado(\_listaDeMuestras);

\_rectasSolucion.Add(resolverSistema(\_listaDeMuestras.Count, sumaX, sumaY, sumaX, sumaXalCuadrado, sumaXporY));

double sumaDistanciasAlCuadrado=sumaDeDistanciasAlCuadrado(\_listaDeMuestras,

\_rectasSolucion.ElementAt(\_rectasSolucion.Count-1));

ParametrosListaDeMuestras \_nuevoParametro = new ParametrosListaDeMuestras

(sumaX,sumaY,sumaXalCuadrado,

sumaXporY, sumaDistanciasAlCuadrado);

return \_nuevoParametro;

}

public Resultado calcularRectas()

{

\_rectasSolucion = new List<Recta>();

List<ParametrosListaDeMuestras> \_listaParametros=new List<ParametrosListaDeMuestras>();

Muestras.ForEach(\_listaDeMuestras=>\_listaParametros.Add(

cargarParametrosDeListaDeMuestras(\_listaDeMuestras)));

return (new Resultado(this,\_listaParametros));

}

}

public class Recta

{

private double \_pendiente, \_ordenada;

public double Pendiente

{

get { return \_pendiente; }

set { \_pendiente = value; }

}

public double Ordenada

{

get { return \_ordenada; }

set { \_ordenada = value; }

}

public Recta(double pendiente, double ordenada)

{

Pendiente=pendiente;

Ordenada=ordenada;

}

public double YPara(double x)

{

return (Pendiente \* x + Ordenada);

}

}

public class Resultado

{

public List<ParametrosListaDeMuestras> \_listaParametrosDeMuestras;

public Combinacion \_combinacion;

public Resultado(Combinacion comb,List<ParametrosListaDeMuestras> \_listaParametros)

{

\_combinacion = comb;

\_listaParametrosDeMuestras = \_listaParametros;

}

public bool errorMenorALoIndicado(double errorMinimo)

{

return (\_listaParametrosDeMuestras.TrueForAll(parametrosDeMuestra=>

parametrosDeMuestra.\_sumaDeDistanciasAlCuadrado<=errorMinimo));

}

}