Instructivo uso de Octave/MATLAB para modelar

Tanto GNU Octave como MATLAB son sistemas de cálculo discreto creados para facilitar la realización de cálculos complejos entre matrices y vectores de datos, así como facilitar diferentes métodos numéricos de análisis de datos. Ambos programas tienen un modo de uso similar y comparten la misma "sintaxis". Se los utiliza de forma análoga a un lenguaje de programación por "scripts". MATLAB es un programa privativo y pago que se puede adquirir e instalar tanto en MS Windows como en GNU/Linux. GNU Octave es software libre y gratuito, y se puede usar también en ambos sistemas operativos

La intención de este instructivo es introducir solo un par de ejemplos básicos que permitan hacer análisis de datos. Para mayor información se proporcionarán links de referencia.

Uso básico de Octave/Matlab

Como en cualquier lenguaje de programación, Octave/Matlab utilizan variables, operadores y funciones. A las variables se les asigna valores con el operador "=", que quiere decir "asignarle tal valor a esta variable" y no tiene nada que ver con el concepto de igualdad matemática.

```
>> a = 2.5
a =   2.5000
>> b = pi
b =   3.1416
>> c = a + b
c =   5.6416
```

Las variables pueden contener vectores o matrices

```
>> posicion=[0, 1, 2, 3, 4, 5]
posicion =
     0     1     2     3     4     5
>> velocidad=[5.5, 4.2, 3.1, 3, 2.9, 2.5]
velocidad =
    5.5000    4.2000    3.1000    3.0000    2.9000    2.5000
```

```
>> matriz= [1,2 ; -3, 5]

matriz =
     1     2
    -3     5
```

En estos casos se pueden acceder a los elementos individuales de una variable vectorial/matricial indicando la posición de un elemento entre paréntesis:

```
>> velocidad(1)
ans =
    5.5000

>> velocidad(5)
ans =
    2.9000
```

```
>> matris(1,2)
ans =
     2

>> matris(2,1)
ans =
    -3
```

Operadores:

+	suma
*	Producto interno/matricial
/	división
^	Potenciación matricial
'	Trasponer matriz

-	resta
.*	Producto elemento a elemento
./	División elemento a elemento
.^	Elevar a una potencia elemento a elemento

Ejemplos:

```
>> a=[1 , -1 ];
>> b=[5 , 3];
```

Creamos dos vectores

```
>> a*b'
ans =
     2
```

Productor interno devuelve un escalar

```
>> a.*b
ans =
     5    -3
```

Producto elemento a elemento devuelve otro vector

Si se le agrega un ";" al final de cada línea no se muestra el resultado de la operación en la pantalla.

Existen formas simplificadas de introducir vectores. Por ejemplo, la sintaxis **`[variable]=[num_inicial]:[paso]:[num_final]`** te permite crear un vector que inicia en [num_inicial] y cada elemento incrementa su valor en [paso] hasta llegar al número [num_final]. Ejemplo

```
>> x=0:0.2:2
x =
         0    0.20    0.40    0.60    0.80    1.00    1.20    1.40    1.60    1.80    2.00
```

Los vectores se pueden combinar para formar vectores mas largos o matrices:

```
>> a=[1, 2, 3] ;
>> b=[4, 5, 6] ;

>> vec=[a, b]
vec =
   1   2   3   4   5   6
```

```
>> mat=[a', b']
mat =
   1   4
   2   5
   3   6
```

En ambos programas se pueden encontrar funciones matemáticas básicas que nos permiten hacer cálculos con números, vectores o matrices:

```
>> x=0:pi/8:pi
x =
         0    0.3927    0.7854    1.1781    1.5708    1.9635    2.3562    2.7489    3.1416

>> y=cos(x)
y =
    1.0000    0.9239    0.7071    0.3827    0.0000   -0.3827   -0.7071   -0.9239   -1.0000


>> a=[0,1; 2, 3];
>> exp(a)
ans =
    1.0000    2.7183
    7.3891   20.0855
```

Esto permite hacer cuentas sofisticadas sobre un vector combinando diferentes funciones y operaciones.

```
>> y=sin(x.^2)./x
y =
       NaN    0.3911    0.7365    0.8347    0.3974   -0.3334   -0.2835    0.3478   -0.1370
```

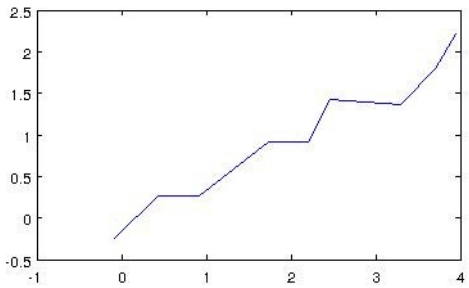
En este ejemplo, para el primer valor de *x* que es 0 hay una división por cero, que nos devuelve el elemento "NaN": *Not a Number*. Hay que tener cuidado con los cálculos realizados para evitar encontrarse con cuentas imposibles como esta.

Creación de gráficos

Es muy fácil graficar un set de datos en Octave/Matlab. Para un gráfico bidimensional clásico sólo hace falta el comando **plot(***vec_x***,** *vec_y***)**, donde *vec_x* es un vector con las coordenadas *x* de cada punto a graficar y *vec_y* será un vector con las coordenadas *y*.

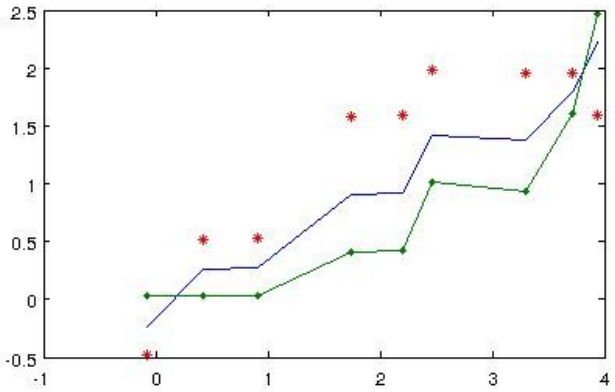
Ejemplo:

```
>> x =[  -0.08, 0.43, 0.91, 1.74, 2.20, 2.46, 3.30, 3.71, 3.94 ];
>> y =[  -0.24, 0.26, 0.27, 0.91, 0.92, 1.42, 1.37, 1.79, 2.22 ];
>> plot(x,y)
```



Se pueden incluir varios gráficos agregando opciones de graficación ente comillas para cada uno:

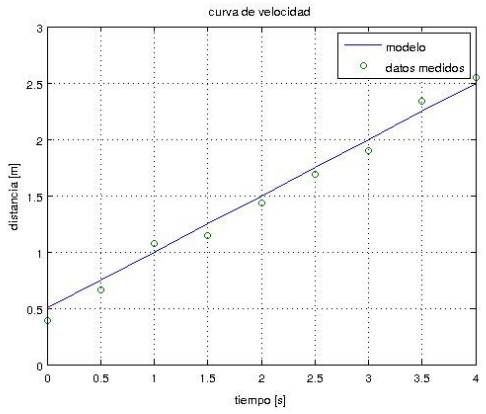
```
>> plot(x, y, '-', x,y.^2/2, '.-', x, 2*sin(y), '*')
```



Ejemplos de opciones	
'-'	Línea recta
'.-'	Línea con puntos
'*-'	Asteriscos con puntos
'o'	Solo círculos
'r-'	Línea roja (red)
'bo'	Círculos azules (blue)

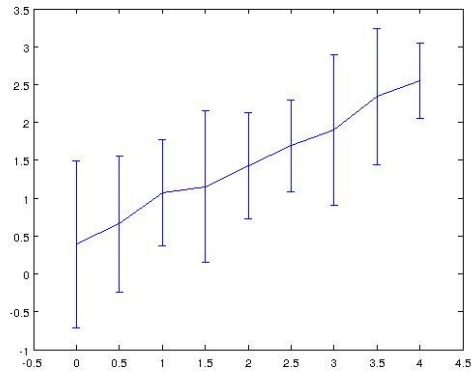
Se le puede dar una presentación más profesional a un gráfico usando otros comandos de formateo:

```
x=0:0.5:4;
y_modelo=x/2+0.5;
y_medidos = [ 0.39, 0.66, 1.07, 1.15, 1.43, 1.69, 1.90, 2.34, 2.55 ];
```




```
plot(x, y_modelo, '-', x, y_medidos, 'o')
grid
legend('modelo','datos medidos')
xlabel('tiempo [s]')
ylabel('distancia [m]')
title('curva de velocidad')
```

La función *errobar* permite graficar un vector con sus barras de error en y:



```
y_err= [1.1, 0.9, 0.7, 1.0, 0.7, 0.6, 1.0, 0.9, 0.5];
errorbar(x,y_medidos,y_err)
```

Los gráficos creados pueden ser exportados para incluirlos luego en un documento o informe. Matlab incluye una interfaz amena en la que solo hace falta hacer clic en Archivo → Guardar y elegir luego el formato. En Octave es necesario ejecutar un comando de texto. Por ejemplo, con estos comandos se exporta el gráfico actual a PNG y luego a EPS (un formato de gráficos vectoriales):

```
>> print ("NOMBRE_DE_ARCHIVO.png", "-color", "-FHelvetica:12", "-dpng")
>> print ("NOMBRE_DE_ARCHIVO.eps", "-color", "-deps")
```

Sistema de archivos

Cuando se ejecutan comandos en Octave/Matlab siempre se está trabajando en alguna carpeta del sistema operativo. Es importante conocer bien la carpeta de trabajo ya que allí exportará el programa sus archivo y buscará archivos para leer. El comando *pwd* nos muestra la carpeta de trabajo y el comando *dir* o *ls* nos muestra el contenido de esa carpeta:

```
>> pwd
ans = /home/lolo/Documentos/matlab
>> ls
datos.mat   g1.jpg  g4.jpg   leasqr.m   modelo2.m
```

Guardar datos y programas

Cuando se trabaja con los datos en el Octave/Matlab se van generando nuevos vectores o matrices que corresponden a datos procesados o al resultado de alguna cuenta. A menudo se quiere guardar esas variables para poder cargarlas luego y continuar con el análisis. Para eso están los comandos *save* y *load*. Para guardar TODAS las variables del sistema de un dado momento en el archivo NOMBRE_DE_ARCHIVO se debe ejecutar:

```
>> save 'NOMBRE_DE_ARCHIVO.mat'
```

También se puede querer guardar solamente las variables var1 y var2 (por ejemplo):

```
>> save 'NOMBRE_DE_ARCHIVO.mat' var1 var2
```

Sea lo que sea que se haya guardado, se lo puede volver a cargar con el comando load:

```
>> load 'NOMBRE_DE_ARCHIVO.mat'
```

Las variables que se carguen desde el archivo van a pisar cualquier otra con el mismo nombre que ya existiese en la memoria.

Por convención, los archivos que guardan variables terminan en *.mat* . A menudo se quiere guardar una secuencia de comandos que se utilizó para procesar ciertos datos o crear un gráfico. Tanto el Octave como el Matlab tiene editores de texto plano donde se pueden escribir los comandos, ejecutarlos en la consola y guardarlos en archivos de texto. Por convención, estos archivos terminan en *.m* .

Funciones para tratamiento de datos

Existen múltiples funciones para realizar toda clase de tareas en Octave y Matlab. La mayoría de las funciones son compartidas entre ambos programas pero existen algunas específicas que difieren levemente de un programa al otro o que se implementan con un nombre distinto. Por mencionar sólo algunas funciones útiles, supongamos que tenemos dos tiras de datos:

```
>> t=0:0.5:4;
>> vel=[0 6 10 14 21 20 22 20 21];
```

Las funciones *sum*, *mean* y *std* nos permiten obtener rápidamente la suma completa, el promedio y la desviación estándar de todos los elementos de un vector:

```
>> sum(vel)
ans =  134
```

```
>> mean(vel)
ans =  14.889
```

```
>> std(vel)
ans =  7.9285
```

La función *diff* nos devuelve un vector con la resta entre elementos consecutivos del vector de entrada (osea la diferencia entre cada elemento y su siguiente).

```
>> diff(vel)
ans =
   6   4   4   7  -1   2  -2   1
>> diff(t)
ans =
0.5   0.5   0.5   0.5   0.5   0.5   0.5   0.5
```

Con ello es fácil realizar una derivada numérica. Hay que notar que el vector resultante tiene un elemento menos que los vectores originales:

```
>> acel=diff(vel) ./ diff(t)
acel =
   12    8    8   14   -2    4   -4    2
```

También se puede realizar la integral numérica con el comando *trapz* (el área debajo de TODA la curva) y *cumtrapz* (el área bajo la curva que se acumula punto a punto, que equivaldría a la primitiva de la función):

```
>> recorrido_total=trapz(t,vel)
recorrido_total =  61.750
>> recorrido=cumtrapz(t,vel)
recorrido =
0.00    1.50    5.50   11.50   20.25  30.50   41.00   51.50   61.75
```

Funciones definidas por el usuario

Se pueden crear funciones definidas por el usuario para ser utilizadas luego en la línea de comandos del Octave o Matlab. Para eso debemos crear un archivo cuyo nombre sea el nombre de la función con la extensión .m y adentro colocar los comandos que definen dicha función. Por ejemplo, si queremos crear una función que se llame "gaussiana" que permita calcular el valor de una gaussiana para un dado *x*, considerando los parámetros de amplitud (*A*) y desviación estándar (*sigma*) debemos crear el archivo *gaussiana.m* y escribir en su contenido:

```
% Calcular el valor de una gaussiana
function [rta]=gaussiana(x,A,sigma)
  rta = A * exp( - x.^2 ./ sigma^2);
end
```

Las líneas que comienzan con **%** no son interpretadas por el programa y sirven para dejar escritos comentarios útiles para el usuario. La expresión [rta] sirve para definir el nombre de la variable de salida de la función. Las variables que están entre paréntesis en la línea de la función son los parámetros de la función que se está definiendo. Una vez guardado como un archivo dentro de la carpeta de trabajo se podrá utilizar la función definida como una función más dentro del programa:

```
>> x=[0 1 2 3];
>> gaussiana(x,1,2)
ans =
   1.00000   0.77880   0.36788   0.10540
```

Todo lo que esta entre el comando *function* y *end* corresponde al script que define la función. Este puede ser tan largo y complejo como se quiera. Al final, la función definida devolverá el valor que este guardado en la variable de salida.

Estructuras de programación

No es el objetivo de este instructivo enseñar a programar. Pero vale la pena mencionar que existen todas las estructuras de programación clásicas en Octave/Matlab. Sólo como ejemplo mencionaremos la del condicional (*if*/*else*) y la de iteración (*for*). Por ejemplo, el condicional siguiente:

```
if x<10
  x=x+1;
else
  x=0;
end
```

En este ejemplo se evalúa si *x* es menor que 10 (la condición que acompaña al *if*). Si es cierto ejecuta `x=x+1;` e incrementa en 1 el valor de *x*. Si es falso, ejecuta lo que aparece después del *else*, que es fijar el valor de *x* en cero.

El bucle *for* nos permite realizar una misma acción para diferentes valores de una variable. Por ejemplo:

```
x=0;
for i=[2 4 8]
  x=x+i^2;
end
```

realizará todo lo que se encuentre entre la línea de *for* y la de *end*, primero para i=2, después para i=4 y así hasta completar el vector.

Ayuda e información sobre otras funciones

Existen miles de funciones ya implementadas en Octave y Matlab para resolver toda clase de problemas. Desde encontrar los ceros de una función matemática, calcular determinantes, autovalores, autovectores, resolver problemas de

optimización/minimización, hacer análisis de Fourier, etc. Se puede hallar una lista de funciones ordenadas en diferentes categorías en esta web:

http://www.mathworks.com/help/matlab/index.html

Si se conoce el nombre de una función pero no se sabe como usarla se puede usar el comando *help* desde la línea de comandos. Por ejemplo, si quiero saber como usar la función *cov* :

```
>> help cov
-- Function File:  cov (X)
 -- Function File:  cov (X, OPT)
 -- Function File:  cov (X, Y)
 -- Function File:  cov (X, Y, OPT)
     Compute the covariance matrix.

     If each row of X and Y is an observation, and each column is a
     variable, then the (I, J)-th entry of `cov (X, Y)' is the
     covariance between the I-th variable in X and the J-th variable in
     Y.

          cov (x) = 1/N-1 * SUM_i (x(i) - mean(x)) * (y(i) - mean(y))

     If called with one argument ...
```

La ayuda debería proporcionar información suficiente para saber aplicar la función y para entender qué es lo que hace concretamente. Puede que algunas funciones de Matlab no especifiquen qué es lo que la función hace concretamente debido a que protegen sus algoritmos para que no se los copien. Las funciones de Octave son todas abiertas y pueden ser estudiadas y modificadas.

En la web se pueden encontrar también galerías armadas como esta:

http://www.mathworks.com/discovery/gallery.html?s_tid=abdoc_plot_1

donde se presentan diferentes tipos de gráficos (2D, 3D, paramétricos, polares, histogramas, etc). Eligiendo cualquiera de ellos muestra un ejemplo de código que explica como hacer ese gráfico.

Introducción al análisis de datos

En las ciencias físicas analizamos los fenómenos proponiendo modelos matemáticos que pueden describirlos cualitativa y cuantitativamente para hacer predicciones luego. El fenómeno es descripto recolectando datos de diferentes variables de interés a las que después se les buscan relaciones. La mayoría de las leyes físicas son relaciones funcionales entre variables de un sistema. Por ende, al desarrollar modelos buscamos encontrar funciones que vinculen las diferentes variables y que sean consistentes con los datos que relevamos.

Dado un conjunto de datos puede haber diferentes funciones que los puedan describir cualitativamente. La pregunta que nos queremos responder es: *Dados dos modelos: ¿cual se corresponde mejor con los datos?*. Buscamos poder evaluar cuantitativamente la fidelidad de un modelo con los datos hallados. Para eso vamos a utilizar algunas herramientas matemáticas del análisis de variables aleatorias y métodos numéricos de análisis. Vamos a introducir esas herramientas para luego ver como las utilizaremos.

Media, varianza y desviación estándar

Supongamos que tenemos un experimento/sistema estable en el cual medimos varias veces el valor de una variable obteniendo diferentes resultados en forma aleatoria. Si no cambiamos ninguna variable del sistema, la aleatoriedad proviene de alguna fuente que no controlamos. Entonces necesitamos hacer algo de análisis para extraer el dato que nos interesa. Por ejemplo, si notamos que la mayoría de los valores obtenidos se acumulan en algún lugar es razonable tomar el promedio de los valores obtenidos como el valor de la variable que queríamos medir. Si tenemos un conjunto de N datos x_i resultantes de las mediciones de una variable A, asumimos que el valor de A es:

$$A \leftarrow \bar{x} \equiv \sum_i \frac{x_i}{N}$$

```
A=sum(x)/length(x);

A=mean(x);
```

Para ponerle una cota de "error" de ese valor es necesario analizar cuanto se dispersan los datos respecto del valor promedio. Esto se puede hacer, por ejemplo, calculando la *desviación estándar* $\sigma = \sqrt{var(x)}$, que es la raíz cuadrada de la varianza:

$$Err_A \leftarrow \sigma_x \equiv \sqrt{\sum_i \frac{(x_i - \bar{x})^2}{N}}$$

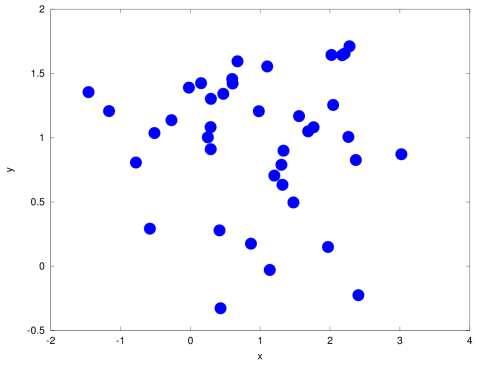
```
varA=sum( (x-mean(x)).^2 )/length(x);
ErrA=sqrt(varA);

ErrA=std(A);
```

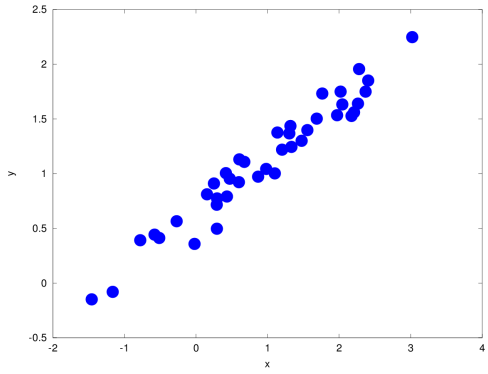
Ojo!. Estrictamente, para poder decir que σ_x nos da información sobre el "error" en la variable A necesitamos conocer la distribución estadística asociada a esa variable. Por ejemplo, para una distribución Normal/Gaussian (una distribución que es muy habitual) el 68% de las veces que se mida A el valor estará entre $\bar{x} - \sigma_x$ y $\bar{x} + \sigma_x$.

Al lado de la definición se presentan dos formas de calcular la varianza en Octave/Matlab. Hay que hacer notar que la función *std* no implementa estrictamente "`sqrt(sum((x-mean(x)).^2 )/length(x))`" (ver *help std*). La diferencia es sutil y tiende a desaparecer para muestras de datos muy grandes. Está relacionada con que "estimar el valor de la varianza" de una variable no es estrictamente igual a la "definición de varianza" en abstracto. El detalle escapa a los fines de esta guía pero se trata con rigurosidad en la materia optativa "Incertezas experimentales y teoría de errores".

Covarianza y correlación

Ahora supongamos que tenemos un experimento en el que medimos varias veces dos variables al mismo tiempo; por ejemplo *x* e *y*. Cada una de estas variables muestran un comportamiento aleatorio, aunque pueden mostrar cierta tendencia en su evolución temporal (Ej: *x* tiende a crecer con el tiempo). Nos interesa saber si comparten información sobre el sistema o si son variables que evolucionan de forma independiente. Si para cada medición i obtuvimos valores x_i y y_i podemos graficar uno respecto del otro. Estos son dos posibles ejemplos de resultados:





En el ejemplo de la izquierda no se puede apreciar una dependencia clara de una variable sobre otra. En el de la derecha, en cambio, parecería haber una relación lineal entre *x* e *y*, un tanto distorsionada. Para poder caracterizar cuantitativamente este hecho se puede calcular la *covarianza* que mide el cambio mutuo entre las variables:

$$cov(x,y) \equiv \sum_i \frac{(x_i - \bar{x})(y_i - \bar{y})}{N}$$

```
cov_xy=sum( (x-mean(x)) .* (y-mean(y))  )/(length(x));
cov_xy=cov(x,y)
```

Nota: *cov* no implementa estrictamente "`sum((x-mean(x)).*(y-mean(y)))/(length(x))`", pero tienden a lo mismo para muchos datos (ver *help cov*).

Si los valores de *x* e *y* tienden a aumentar y disminuir conjuntamente la covarianza será un número positivo. Si cuando una de las variables aumenta la otra disminuye, entonces será negativa. Si los comportamientos son independientes, tenderá a ser cero. Vale notar que la "auto-covarianza" es la varianza: $cov(x,x) = var(x) = \sigma_x^2$. De los ejemplos anteriores, el de la izquierda tiene $cov(x,y) = -0.015867$ y el de la derecha $cov(x,y) = 0.56881$.

La covarianza nos da una idea de si dos variables comparten información de algún modo, pero su valor absoluto es poco útil, pues varia mucho entre diferentes conjuntos de datos. Para evitar este problema se puede usar la *correlación* (*corr*) o *coeficiente de correlacion de Pearson* (*R*), que son lo mismo pero suelen denominarse con diferentes letras.

$$corr(x,y) \equiv \frac{cov(x,y)}{\sigma_x \sigma_y}$$

```
R=cov(x,y)/ std(x) / std(y);
R=corr(x,y);
```

Se puede demostrar fácilmente que si *x* e *y* tienen una dependencia lineal del tipo $y_i = A \cdot x_i + B$ entonces la correlación da:

$$R = corr(x,y) \equiv \frac{cov(x, Ax+B)}{\sigma_x \sigma_y} = \frac{A\, cov(x,x)}{\sigma_x |A| \sigma_x} = \frac{A}{|A|} = \pm 1$$

Cuando $R = 1$ se puede decir que hay una perfecta correlación lineal entre las variables y cuando $R = -1$ hay una perfecta anticorrelación. De los ejemplos de arriba, el de la izquierda tiene $R = -0.027934$ y el de la derecha $R = 0.97180$.

Cuando se tienen los conjuntos de valores medidos x_i e y_i y se realiza un **ajuste lineal** de los datos con el modelo $f_i = A\, x_i + B$, el cuadrado de la correlación entre los datos medidos y los valores del ajuste $R^2 = corr(y,f)^2$ es denominado *coeficiente de determinación*. R^2 puede ser interpretado como "el porcentaje de la varianza de y que se puede explicar a partir de *x*". Muchos programas de análisis de datos ofrecen funciones para hacer una **regresión lineal** de los datos y reportan el valor de R^2 entre los resultados. Para **modelos no lineales** el

coeficiente de determinación no coincide con esta definición y esta interpretación deja de ser estrictamente válida.

R^2 es una estimación buena sobre "cuan bien son explicados los datos por un modelo lineal", pero no alcanza por sí solo para saber si una particular elección de parámetros A y B hacen un buen ajuste. Para ello hay que tener en cuenta otras consideraciones que se comentan a continuación.

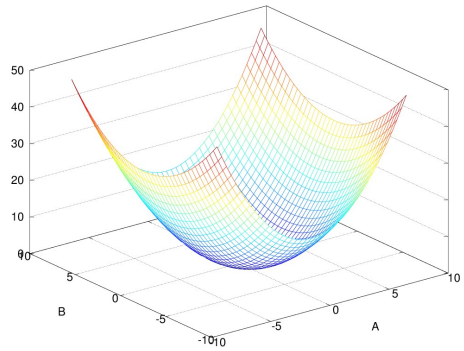
Ajustes de datos mediante cuadrados mínimos

El método de cuadrados mínimos es uno de los más usados para ajustar los parámetros de una función que intenta modelar un conjunto de datos. El método consiste en lo siguiente:

Se tienen N pares de valores de mediciones asociadas a un fenómeno: x_i e y_i . Se tiene una función F que depende de alguna cantidad de parámetros (propongamos por ejemplo que depende de A y B) y se quieren encontrar los parámetros A y B tal que los valores $f_i = F_{A,B}(x_i)$ aproximen lo mejor posible a y_i . El método de cuadrados mínimos propone minimizar la siguiente cantidad, denominada *chi-cuadrado*:

$$\chi^2_{datos}(A,B) = \sum_i (y_i - f_i)^2 = \sum_i (y_i - F_{A,B}(x_i))^2$$

Osea, minimizar la suma cuadrática de los "residuos", que son las diferencias entre los valores medidos y los valores que predice el modelo $r_i = y_i - f_i$. Eso reduce el problema de "buscar los parámetros de un modelo que ajuste los datos" a un problema de minimización. Hay que notar que la función χ^2 se construye a partir de los N datos medidos pero solo tiene como variables a los parámetros (es este caso, A y B). Por ejemplo, si el modelo es una recta $F_{A,B}(x) = A\,x + B$ la función χ^2 sera un paraboloide en 3D y se deberá hallar las coordenadas del mínimo.



Para el caso lineal existen métodos muy eficientes para hallar los parámetros que corresponden al mínimo global del paraboloide. Pero cuando se utilizan modelos no lineales la función χ^2 puede resultar más compleja. Si se usan dos parámetros, χ^2 puede ser una superficie con varios mínimos locales y no es trivial hallar el mínimo global. No nos vamos a detener a explicar cómo funcionan los diferentes algoritmos de minimización, tema que se trata extensamente en la materia Elementos de Calculo Numérico. Simplemente es necesario ser conscientes de cómo funciona el método. En general, se parte de un par de parámetros iniciales y se recorre el camino de mayor pendiente hacia el mínimo. Encontrar el mínimo global dependerá de la elección de esos parámetros iniciales.

Una vez hallados los valores de los parámetros A y B, el valor de χ^2 puede usarse para estimar la bondad de un ajuste. Si se comparan varios modelos sobre un mismo conjunto de datos, el que tenga menor χ^2 será, en principio, un mejor ajuste. Sin embargo, χ^2 no puede usarse para saber si un modelo en si mismo es un buen modelo, pues su valor absoluto no tiene un significado intrínseco.

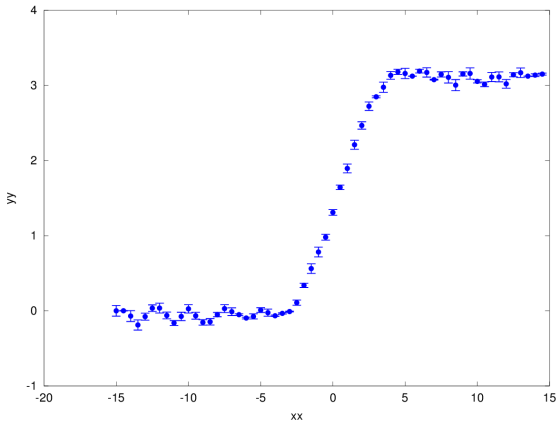
Si se quiere tener en cuenta que algunos datos fueron medidos con mayor precisión que otros se puede construir la función χ^2_w (*wighted chi-squared*) utilizando los datos del error de cada medición δy_i :

$$\chi^2_{weighted}(A,B) = \sum_i \frac{(y_i - f_i)^2}{\delta y_i^2}$$

A los efectos prácticos de esta guía, se presentará a continuación un ejemplo de como ajustar una serie de datos con dos modelos diferentes usando una implementación de cuadrados mínimos para Octave / Matlab.

Ejemplo practico: ajuste y comparación de modelos en Octave/Matlab

Este ejercicio puede encontrarse en la página de la materia con todos sus archivos disponibles. Vamos a usar de ejemplo los datos del archivo *datos.mat* , guardados en las variables *xx*, *yy* y *errores*:



Modelo *error_function*:

$$F_{[A,w,x_0,y_0]}(x) = A \ \ erf\left(\frac{x-x_0}{w}\right)+y_0$$

Modelo arco_tangente:

$$F_{[A,w,x_0,y_0]}(x) = A \ atan\left(\frac{x-x_0}{w}\right)+y_0$$

Proponemos dos posibles modelos que pueden aproximar cualitativamente esos datos: uno basado en la función *error_function* (que es la primitiva de una campana gaussiana) y otra basadao en el *arco tangente* (que es la primitiva de una campana lorentziana).

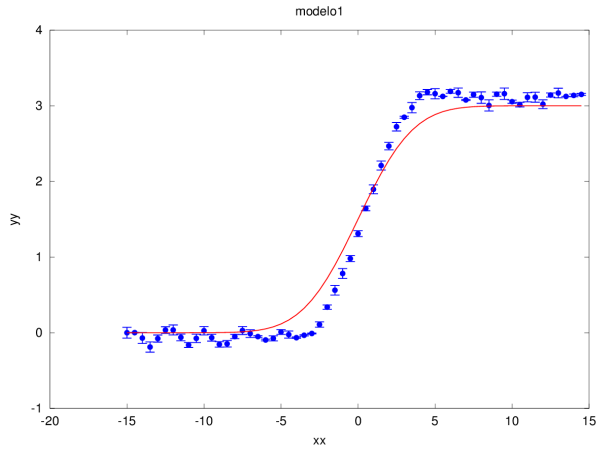
Lo primero que necesitamos hacer es construir dos funciones, que llamaremos *modelo1* y *modelo2*, que nos permitan predecir valores de *yy* a partir de los valores de *xx* utilizando los parámetros que les demos:

Archivo modelo1.m

```
function [y]=modelo1(x,par)
    y=par(1)*erf((x-par(2))/par(3))+par(4);
end
```

Modo de uso:

```
A=1.5;
x0=0;
w=4;
y0=1.5;
parametros1=[A, x0, w, y0];
ff1=modelo1(xx,parametros1);
```

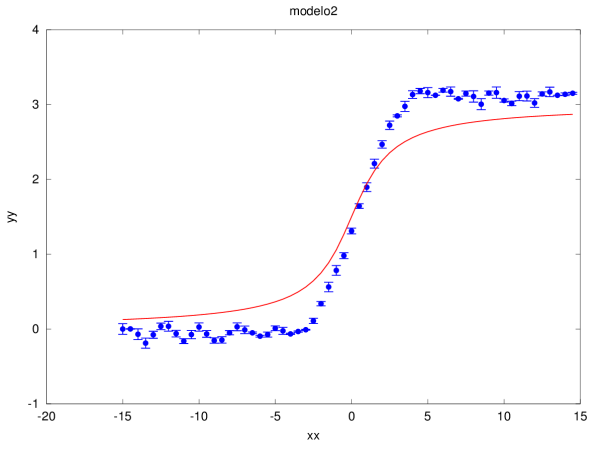


Archivo modelo2.m

```
function [y]=modelo2(x,par)
    y=par(1)*atan((x-par(2))/par(3))/pi*2+par(4);
end
```

Modo de uso:

```
A=1.5;
x0=0;
w=2;
y0=1.5;
parametros2=[A, x0, w, y0];
ff2=modelo2(xx,parametros2);
```



Con las funciones armadas se pueden probar “a mano” parámetros razonables para que la curva de nuestros modelos se parezcan a los datos. Estos parámetros estimativos nos servirán de punto de partida para aplicar el método de cuadrados mínimos.

Luego, vamos a utilizar alguna de las funciones que existen para optimizar los parámetros por cuadrados mínimos. En

particular, usaremos la función *leasqr* (incluida en los archivos de la página de la materia), que tiene la siguiente sintaxis:

```
[ff, parametros, ok, iter, corP, covP]
              =leasqr(x, y, parameros_iniciales, 'funcion_modelo',.0001,20,pesos);
```

donde las variables de entrada son los vectores *x* e *y* de los datos, un vector *parameros_iniciales* con los parámetros de partida para aplicar cuadrados mínimos y el nombre de la función que implementa el modelo que se va a fitear (solo el nombre entre comillas). Los valores 0.0001 y 20 corresponden a parámetros de tolerancia y de numero de iteraciones máximas a realizar, se pueden dejar esos valores que son razonables. La variable *pesos* es un vector con los pesos estadísticos para construir χ^2_w y se la puede definir como $pesos = 1/\delta y$.

Las variables de salida son: *parametros*, con los valores óptimos hallados para los parámetros del modelo, *ff*, con los valores que predice el modelo para cada valor de *x*, la variable *ok* que nos dice si se hallaron los parámetros óptimos o no, la variable *iter* que nos dice la cantidad de iteraciones realizadas en el método de cuadrados mínimos, y por ultimo *corrP* y *covP*, que son las matrices de correlación y de covarianza de los parámetros hallados.

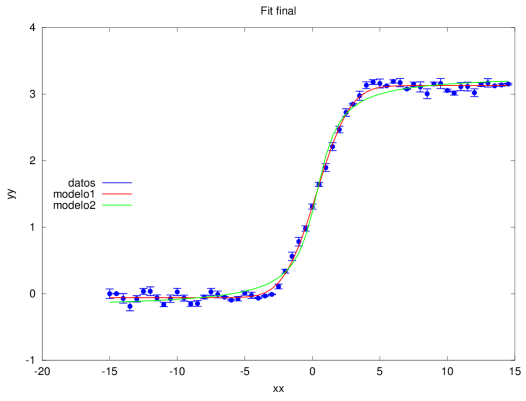
Ejecutando *leasqr* para cada modelo:

```
[f1,p1,cvg1,iter1,corp1,covp1]=leasqr(xx,yyi,parametros1,'modelo1',.0001,20,pesos);

[f2,p2,cvg2,iter2,corp2,covp2]=leasqr(xx,yyi,parametros2,'modelo2',.0001,20,pesos);
```

Se obtienen los siguientes resultados:

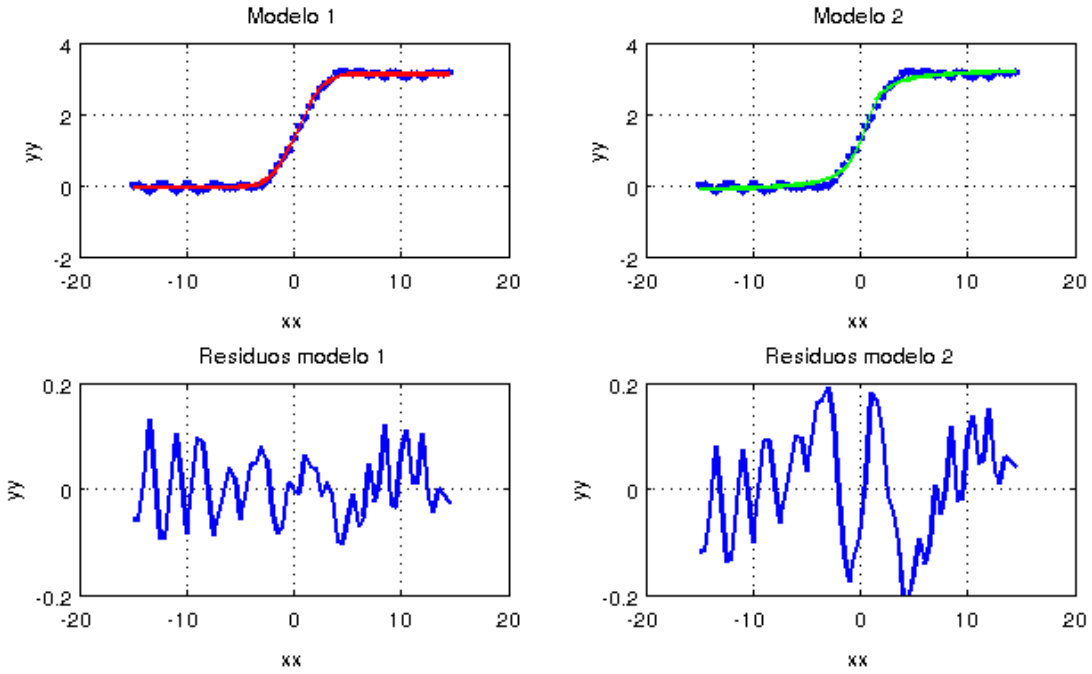
```
parametros1 =
    1.5000         0    4.0000    1.5000
p1 =
    1.5911    0.3468    2.7125    1.5336

chi2_modelo1 = sum( (yy-f1).^2 )
             =   0.2112
R2_modelo1   = corr(yy,f1)^2
             =   0.9984
```

```
parametros2 =
    1.5000         0    2.0000    1.5000
p2 =
    1.7470    0.3613    1.2205    1.5405

chi2_modelo2 = sum( (yy-f2).^2 )
             =   0.6381
R2_modelo2   = corr(yy,f2)^2
             =   0.9951
```



Se puede apreciar que el modelo 1 obtuvo un valor de χ^2 inferior , lo que implica un mejor ajuste, y que el coeficiente de determinación también es mejor, lo que implica que el modelo es más adecuado para describir los datos. Esta definición de R^2 no suele ser la que utilizan los programas de cuadrados mínimos, pero coincide en el caso de una regresión lineal. Cuando tenemos un modelo no lineal (como en este caso) usar esta definición nos facilita comprender su significado. R^2 Nos dice qué tan bien correlacionan los datos del problema y_i con las predicciones de nuestro modelo f_i . Cuanto más cercano a 1 es R^2 quiere decir que más lineal es la relación que hay entre y_i y f_i .

R^2 no cambia si nuestro modelo erra por un factor multiplicativo o por la suma de una constante. Pero sí se aleja mucho de 1 si la relación de los datos con el modelo no es lineal, lo que implica que la relación funcional planteada no es adecuada para modelar el problema. Esto se pone en evidencia si se grafican los residuos, porque en vez de parecer "ruido" van a mostrar alguna estructura, relacionada con la parte del fenómeno que no estamos modelando:



Por último, los programas que hacen ajuste de funciones generalmente nos entregan más información que a menudo no sabemos utilizar. Explicar de donde surge y cómo se calcula esa información escapa a los fines de este instructivo, pero vamos a comentar un poco cómo se la puede utilizar.

Uno de los datos que suele suministrar un programa de fiteo es la *matriz de covarianza de los parametros*, en este caso *covp1* y *covp2*. Para el modelo , *covp1* nos muestra la covarianza entre los parámetros suministrados [A, x0, w, y0] (donde el orden definido en la función *modelo1* es importante):

```
covp1 =
   0.00010423  -0.00000384   0.00036114  -0.00000077
  -0.00000384   0.00185711  -0.00030509   0.00019907
   0.00036113  -0.00030508   0.00744511   0.00003590
  -0.00000077   0.00019907   0.00003589   0.00009574
```

La primera fila/columna corresponde a *A*, la segunda a *x0*, la tercera a *w* y la cuarta a *y0*. Como la covarianza de una variable con sí misma es la varianza, los elementos de la diagonal de esa matriz son la varianza de cada uno de los parámetros. Por ende, la raiz de cada uno de esos valores es la *desviación estándar de cada parámetro*:

```
>> p1
p1 =
    1.5911    0.3468    2.7125    1.5336

>> sqrt(diag(covp1))
ans =
    0.0102    0.0431    0.0863    0.0098
```

La desviación estándar nos puede servir para ponerles cotas de error a las parámetros hallados.

Por último, la matriz de correlación (en este caso *corp1* y *corp2*) nos da información sobre cuan dependiente es un parámetro de otro en nuestro modelo. Si dos parámetros llegan a tener una correlación alta, eso quiere decir que probablemente se puede reemplazar uno de ellos por una función de otro.

Por ejemplo, podríamos tener un modelo con 4 parámetros [a, b, c , d] y hallar que *a* y *d* tienen alta correlación. Si para varias series de datos el fiteo nos da siempre *a~d* , entonces lo más probable es que estemos usando parámetros de más para modelar. En el fondo debemos reemplazar nuestro modelo por uno que incluya solo 3 parámetros: [a, b, c].