Octave a lo FIUBA

Curso rápido de GNU Octave - Parte 1

Losotro’

March 12, 2016

FIUBA

Temario

1. Octave como calculadora
2. Trabajando con matrices
3. Gráficos Bi-dimensonales
4. Gráficos Tridimensonales
5. Tarea integradora

Octave como calculadora

Calculadora simple

Octave puede usarse como calculadora. Simplemente podemos escribir operaciones matemáticas y ver la respuesta.

Calculadora simple

Octave puede usarse como calculadora. Simplemente podemos escribir operaciones matemáticas y ver la respuesta.

Notación

Las lineas que comienzan con denotan los comandos que *nosotros* escribimos en el programa, las otras son la respuesta que devuelve.

Lo que sigue a un símbolo % es un comentario.

Probemos esto

```
> 3+2
5
> 3*5+1.1  % El separador decimal es el .
11.1
> sqrt(16) % sqrt es abreviatura de SQuare RooT
4
> i^2
-1
> (1+3i)*(2-3i/2)
> 1/2i
```

Numeros complejos

Octave maneja cómodamente números complejos. La unidad imaginaria es indistintamente `i` o bien `j` (esta última notación se usa mucho por los electricistas y electrónicos, los matemáticos parecen preferir la primera).

Buenas prácticas

Por desgracia, `i` y `j` son comúnmente usadas como nombres de variables. Para evitar problemas, siempre ponga un número adelante de la unidad imaginaria. Es decir, en lugar de:

```
> 1+i
```

Escriba:

```
> 1+1i
```

Forma Binómica y polar

Para obtener la parte real e imaginaria, y el módulo y ángulo de un complejo, tenemos estas funciones:

```
> real(1-2 i )
1
> imag(1-2 i )
-2
> abs(1-2 i )
> angle(1-2 i )
> conj(1-2 i )
1+2 i
```

Usando variables

Una necesidad básica es guardar la información para volver a usarla después, como las memorias de las calculadoras. En Octave podemos hacer eso usando *variables*.

Variables

Para *asignar* un valor a una variable, use el operador `=`

Ejemplo: `a=3` define una variable a y le asigna el valor numérico tres.

Probemos esto

```
> a = 3*7 + 2 - 7^2
> 3*a + 17
> 3a + 17 % Esto no funciona, falta el simbolo de por
> c = 2*a + 3
> d = 1000*a;  % El ; hace que no diga el resultado
> d  % Para mostrar el valor de una variable
> disp(d)  % Esto hace lo mismo per es más elegante (?)
```

Funciones Básicas

GNU Octave posee un conjunto de funciones y operaciones matemáticas básicas ya implementadas. Pueden descargarse más desde OctaveForge[1]. Algunas son:

Redondeo
- floor()
- ceil()
- round()

Trigonometría
- sin()
- sind()
- cos()
- tanh()

Complejos
- abs()
- angle()
- real()
- imag()

Logaritmos
- log()
- log10()
- log2()
- exp()

Para saber qué hacen use el comando help o el comando doc seguido del nombre de la función. Por ejemplo: `help sind` o `doc angle`. Si no, Googleé :)

[1] Visitar http://octave.sourceforge.net/

Probemos esto

```
> abs(-3)
> abs(1+1i)+100*angle(1+1i)
> exp(2-3j)
> sin(1+2j)
> exp(2)*( cos(-3) + 1i*sin(-3) )
> sin(pi)
```

Notemos esta pequeña diferencia:

```
> sin(pi)
> sind(180)
```

Las funciones trigonométricas vienen por defecto en radianes; `sind`, `cosd` y `tand` usan grados sexagesimales.

Probemos esto

Ahora vamos a calcular el seno de 30 grados pasando primero a mano a radianes y usando luego `sin()`. Vamos a usar variables para la claridad y reutilizabilidad del código.

```
> angulo_grados = 30;
> angulo_radianes = angulo_grados * pi / 180;
> sin(angulo_radianes)
```

Y vemos que da lo mismo que

```
> sind(angulo_grados)
```

Sobre los nombres de variables

Use nombres descriptivos

Tenga en cuenta que alguien más va a querer leer su código. Haga que sea lo más claro y entendible posible; sepa que aún esa persona puede ser usted mismo en el futuro ;)

Evite abreviaciones y use siempre un mismo estilo (por ejemplo, `separar_con_guiones` o `separarConMayusculas`). Elija español o inglés y *sea consistente* en lo posible.

Es preferible (generalmente) separar las ecuaciones complicadas en varias partes en lugar de tener expresiones kilométricas inentendibles

Trabajando con matrices

Matrices y Vectores

El tipo de datos básico en Octave son las matrices. Los escalares son un caso particular de matrices de 1×1 y los vectores lo son en el caso de $1 \times n$ o $n \times 1$.

Muchas de las funciones que se usan para números reales funcionan sobre vectores y matrices también.

Definiendo matrices

Para definir una matriz, se utilizan los *corchetes*: `[ ]`, las columnas se separan con *espacio* o `,` y las filas se separan con una *nueva línea* (un "enter") o un `;`

Por ejemplo, la matriz:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Se escribe en Octave como:

```
> A = [1 2; 3 4]
```

O bien, usando un salto de línea en lugar del punto y coma:

```
> A = [1 2
       3 4]
```

Vectores

Y para el caso de vectores es lo mismo:

$$\mathbf{v} = \begin{bmatrix} 1 & 1 & 2 \end{bmatrix}$$

```
> v = [1 1 2]
```

$$\mathbf{v} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$$

```
> v = [1; 1; 2]
```

Transpuesta y Transpuesta conjugada (hermítica)

Para obtener la transpuesta hermítica de una matriz (o vector), se utiliza el operador `'` (*apóstrofo*[2] *simple*).

Para transponer una matriz (o vector) sin conjugar, se debe utilizar el operador `.'`

Ejemplo:

```
> A = [1-1j 2; 3 4]
> A_h = A'
1 + 1j    3
2         4
> A_t = A.'
1 - 1j    3
2         4
```

[2]Apóstrofe: Insulto que provoca y ofende.| Figura retórica que consiste en interrumpir el discurso para dirigirse con vehemencia a otra persona, generalmente con un tono patético o de lamento.

¿Vectores filas o columna?

A la mayoría de las funciones de Octave les da lo mismo si un vector es fila o columna, pero ¿cuándo usar uno u otro?

Matemáticamente, estamos acostumbrados a los vectores columna (porque hacemos multiplicaciones matriciales de la forma $\mathbf{y} = \mathbf{A}\mathbf{x}$ y no $\mathbf{y} = \mathbf{x}\mathbf{A}$).

Pero en Octave parece que es más cómodo –y usual– tener vectores fila:

```
> x = [1 -2 0 3]
```

¿Vectores filas o columnas?

Pensemos en una tabla de valores donde se anota la evolución de distintas magnitudes en el tiempo: en cada columna hay distintas mediciones de un mismo dato. Ésta misma idea es en Octave: cada columna representa un distinto tipo de dato.

Si tenemos un vector que registra –por ejemplo– la caída de tensión en una resistencia en función del tiempo o la temperatura, debería ser un vector columna. Éste debería ir acompañado –seguramente– por otro vector columna que indique los instantes de tiempo o valores de temperatura correspondientes a cada medición del primero.

¿Vectores filas o columnas?

Estos dos vectores columna pueden acomodarse cómodamente en una matriz, por ejemplo:

```
mediciones = [1     1.23  % tiempo (seg), tensión (volt)
              1.1   1.27
              1.3   1.26
              1.5   1.22
              1.6   1.25
              1.9   1.24
              ];
```

Truquito

Esto puede ser cómodo para escribir *vectores columna* largos, evitando el uso de muchos **;** o saltos de líneas.

```
> x = [1 1 3 -2 4 0 -2 -4 9 0.1 2 -4 -2 8].'
```

Multiplicando matrices I

Si las dimensiones de las matrices (o vectores) concuerdan, se pueden multiplicar usando el operador *

Ejemplo:

```
> A = [1 3 5; 2 1 -1];  % Una matriz de 2x3
> b = [1 2 1].';        % Esto es un vector fila de 3x1
> A*b
12
3
```

Multiplicando matrices I

De la misma forma, podemos hacer potencias de matrices cuadradas, con la definición de $\mathsf{A}^n = \underbrace{\mathsf{AAA}\cdots\mathsf{A}}_{n\text{veces}}$

```
> A = [1 0 1; 0 1 0; 1 1 1];
> A*A*A
> A^3
```

Operaciones elemento a elemento

Supongamos que medidmos la tensión sobre un resistor de 8Ω y tenemos esa información en un vector columna V. Queremos obtener la potencia disipada, dada por la ecuación $P = V^2/R$ ¿Cómo podemos hacer eso?

Si realizamos la operación `V^2` en Octave, devuelve error, ya que V no es una matriz cuadrada. Pero en realidad, lo que queremos hacer es elevar *cada componente* al cuadrado, y no al vector por así decirlo.

Operaciones elemento a elemento

Para indicarle a Octave que una operación debe realizarse elemento a elemento, se antepone un punto. Por ejemplo

- Multiplicación `.*`
- División `./`
- Potencia `.^`

En nuestro caso, tendríamos que hacer:

```
> V = [10.4 7.5 12.4 9.2 7.3].'; % Datos de ejemplo
> R = 8;
> P = V.^2 ./ R
```

Operaciones elemento a elemento

Ejemplo 2: Supongamos que tenemos un vector con frecuencias angulares ω, queremos convertir esos datos al período correspondiente T según la ecuación $T = 2\pi/\omega$, tenemos entonces:

```
> w = [132.43  22.54  563.01].'; % Datos
> T = 2*pi ./ w
```

Broadcasting[3]

¡Esto no es MATLAB!

Ésta es una característica que MATLAB no poseé –y que seguramente no implementará por retrocompatibilidad– así que hay que tener cuidado si se quiere interoperabilidad.

Octave permite multiplicar matrices si sus dimesiones son *parecidas* pero no compatibles formalmente. Por ejemplo, se puede multiplicar una matriz de $n \times 3$ por un vector de 1×3 o 3×1. Lo que hace es multiplicar cada columna por el valor que dice el vector.

Ésta no es una funcionalidad sumamente empleada, pero tenía que contárselas. A veces es útil –depende de cuán loco esté cada uno–. Al menos para que sepan por qué a veces no explota algo que no tendría que funcionar.

[3] Es algo como “Ajuste automático de ancho” en español.

Índices en matrices

Sea A una matriz de $n \times m$. Por ejemplo:

```
> A = [ 1  2  3  4
       -1  0  1  0
        3  9  8 -1];
```

Podemos obtener el elemento de la fila `i` y columna `j` ($1 \leq i \leq n$, $1 \leq j \leq m$) de la forma `A(i,j)`

Ejemplo:

```
> A(3,2)
9
```

Los índices empiezan en 1

A diferencia de lenguajes como C o Java, los índices empiezan por 1 y deben ser positivos.

Índices en matrices - submatrices

Se puede extraer una submatriz especificando un *rango* de valores, denotados por un `:` de la forma `inicio:fin`

Ejemplo, sea:

```
> A = [ 1 2 3 4
       -1 0 1 0
        3 9 8 -1];
```

Luego:

```
> A(1:2, 2:3)
2   3
0   1
```

Índices en matrices - submatrices

Una forma cómoda para especificar que queremos todos los valores de filas o columnas es poner simplemente un `:` en la dimensión que no queremos poner restricciones:

```
> A = [ 1 2 3 4
       -1 0 1 0
        3 9 8 -1];

> A(:,2)
2
0
9

> A(1:2, :)
 1 2 3 4
-1 0 1 0
```

Índices en matrices - end

Muchas veces no tenemos en mente cual es el largo o ancho de una matriz, para ello, podemos valernos de la palabra reservada **end**, que al verla Octave la convierte automáticamente por el tamaño de la dimensión en la cual se use.

```
> A = [ 1  2  3  4
       -1  0  1  0
        3  9  8  -1];
> A(1, end)
4
> A(end-2, end-1)
3
> A(:, end-1)
3
1
8
```

Índices en matrices - end

La palabra **end** se puede usar dentro de cualquier expresión, pero hay que tener cuidado que para que sea un índice válido, debe ser un entero. Por ejemplo

```
> v = [-5 -4 -3 -2 -1 0 1 2 3].';

> v(end/2) % end/2 da 4.5 y esto no anda
> v( floor(end/2) ) % redondeamos para abajo
-2
```

Rangos

La funcionalidad de los dos puntos es generar un rango.

Un rango es de la forma: `inicio:paso:fin` O bien: `inicio:fin` entendiéndose el paso igual a 1 en este caso.

Por ejemplo:

```
> 1:5
1 2 3 4 5
> 1:2:10
1 3 5 7 9
> 15:-1:10
15 14 13 12 11 10
```

Como vemos, el paso puede ser positivo o negativo.

Rangos

Los rangos se pueden utilizar para crear vectores. Notemos que podemos usar numeros no enteros:

Por ejemplo:

```
> v = 1:0.1:1.5
1  1.1  1.2  1.3  1.4  1.5

> 2.*v
2  2.2  2.4  2.6  2.8  3

> 10.^v  % Útil para escalas exponenciales
10.0000  12.5893  15.8489  19.9526  25.1189  31.6228
```

Invirtiendo un vector

¿Cómo puedo hacer para “dar vuelta” un vector? Visto lo anterior es muy sencillo:

```
> v = [0.1 0.2 0.3 0.4 0.5]; % Me cansé de los enteros
> v_rev = v(end:-1:1)
0.5 0.4 0.3 0.2 0.1
```

Y puedo obtener solo las coordenadas pares e impares:

```
> x = [1 0 3 0 9 0 -10];
> x_par = x(2:2:end)
1 3 9 -10
> x_impar = x(1:2:end)
0 0 0
```

Para matrices es *exactamente* igual. (Tarea: Jugar con ese caso).

Seguimos indexando

Podemos decir que un rango es un vector fila[4]. De forma dual, podemos usar vectores para extraer sólo determinados indices:

```
> A = [  1  2  3  4
        −1  0  1  0
         3  9  8  −1];

> A([1  3], 1:2)
1  2
3  9
```

[4] Although a range constant specifies a row vector, Octave does not normally convert range constants to vectors unless it is necessary to do so. This allows you to write a constant like ‘1 : 10000’ without using 80,000 bytes of storage on a typical 32-bit workstation. –https://www.gnu.org/software/octave/doc/interpreter/Ranges.html

Seguimos indexando

Sea

```
> A = [ 1  2  3  4
       −1  0  1  0
        3  9  8  −1];
```

¿Qué hace esto?

```
> A([1  1  2  1], :)
```

Seguimos indexando

Sea

```
> A = [ 1  2  3  4
       −1  0  1  0
        3  9  8  −1];
```

¿Qué hace esto?

```
> A([1  1  2  1], :)  % Repite los elementos
   1   2   3   4
   1   2   3   4
  −1   0   1   0
   1   2   3   4
```

Técnicas avanzadas de indexado[5]

En lugar de un rango, se puede poner una condición. Por ejemplo:

```
> v = [8.1  9.0  1.2  9.1  6.3  0.9  2.7];

> v(v>3)
8.1  9.0  9.1  6.3

> v(v.^2 < 2*v)
1.2  0.9
```

[5]Ver: https://www.gnu.org/software/octave/doc/interpreter/Advanced-Indexing.html

Técnicas avanzadas de indexado

Si disponemos de dos vectores del mismo largo, podemos usar uno como condición del otro. Ejemplo:

```
> t = 0:0.1:10; % Un par de datos, cualquier cosa.
> x = 10 - 3.*t;

> x(t>9.5)
-18.8   -19.1   -19.4   -19.7   -20.0
```

Técnicas avanzadas de indexado

Podemos usar más de una condición, empleando los operadores lógicos elemento a elemento[6]:

- `&` Operador "y" lógico (*and*)
- `|` Operador "o" lógico (*or*)
- `~` Operador "no" lógico (*not*)

```
> t = 0:0.1:10; % Un par de datos, cualquier cosa.
> x = 10 - 3.*t;

> x( (t>2 & t<2.5) | t>9.7 )
3.7  3.4  3.1  2.8  -19.4  -19.7  -20.0
```

[6] Los operadores && y || operan sobre escalares y devuelven escalares

Técnicas avanzadas de indexado

¿Y si quero saber qué rango de índices del vector `t` son los que cumplen que $2.2 < t < 3$?

Para eso existe el comando `find()`. (Hacer `help find` cualquie cosa)

```
> t = 0:0.1:10; % Un par de datos, cualquier cosa.
> x = 10 - 3.*t;

> indices = find ( t>2.2 & t<3)
24    25    26    27    28    29    30
```

Y ahora ya sabemos cuales son esos indices. Podemos ahora usar esos índices de esta forma:

```
> indices = find ( t>2.2 & t<3)
> x(indices)
```

Técnicas avanzadas de indexado

Si bien estas dos expresiones hacen lo mismo, la segunda es mucho más recomendable (menos costosa computacionalmente):

```
> x( find ( t>2.2 & t<3) )  % No; mueren gatitos :(

> x(  t>2.2 & t<3 )         % Sí :D
```

Técnicas avanzadas de indexado

En este caso, sirve tener los índices. Por ejemplo:

Supongamos que tenemos esta curva plana, parametrizada en t. Queremos obtener los valores de x y t para los cuales $y < 3$:

```
> t = 0:0.1:10;
> x = 10 - 3.*t;
> y = 10*t - 0.5*t.^2;

> indices = find(y<3);
> x_principio = x(indices);
> t_principio = t(indices);
```

Ah, para ver un gráfico podemos hacer: `plot(x,y)`. Simplemente une los puntos (x,y) con rectas. `x` e `y` deben ser vectores de igual dimensión –fila o columna–.

¿Qué sigue?

Próxima diapositiva

En la próxima diapositiva vamos a ver cómo definir funciones, realizar gráficos y trazar curvas en 2D y 3D y esas cosas divertidas como resolver ecuaciones diferenciales e integrar.

¿Y después?

Nos vamos al pasto con *cell arrays* y *strings*; vamos a analizar datos automáticamente y hacer gráficos muy copados.

A la misma batihora y por el mismo baticanal.

¿Qué sigue?

Próxima diapositiva

En la próxima diapositiva vamos a ver cómo definir funciones, realizar gráficos y trazar curvas en 2D y 3D y esas cosas divertidas como resolver ecuaciones diferenciales e integrar.

Gráficos Bi-dimensonales

Gráficos 2D plot()

La función `plot()` se encarga de realizar gráficos en ejes cartesianos. Puede dibujar más de una curva en el mismo gráfico y permite modificar algunas propiedades de diseño como los colores y estilos de las líneas. `plot()` puede tomar una cantidad de argumentos variable e interpreta a los vectores o matrices como datos de las curvas y a las cadenas de texto como las propiedades de diseño. La forma general de los argumentos de `plot()` son de la forma siguiente.

Función plot - argumentos

plot(x1 , y1 , 'propiedad1' , 'valor1' , 'propiedad2' , 'valor2' , x2 , y2 , 'propiedad3' , 'valor3' , ...)

Cada valor hace referencia a la propiedad inmediatamente anterior, y cada propiedad modifica el par de datos x , y anterior.

Gráficos 2D plot()

Probemos algunos ejemplos para ver que hace:

```
> t = 0:0.1:10; % Vector fila de 101 elementos.
> x = 10 - 3.*t; % También fila de 101 elementos.

> plot( x ); % Grafica los puntos ( i , x(i) )
> plot ( t , x); % Grafica los puntos ( t(i) , x(i) )
> plot (t , x , 'color' , 'r' , 'linestyle', ':')
```

El último gráfico modificó el color de la línea a rojo y la dibujó punteada. Las propiedades que se pueden modificar para cada curva son: “linestyle”, “linewidth”, “color”, “marker”, “markersize”, “markeredgecolor”, y “markerfacecolor”.

Para más info sobre las propiedades poner `"help plot"` en la consola de Octave.

Gráficos 2D plot()

Por último, y porque sin etiquetas en los ejes y título del gráfico el TP rebota...

```
> plot ( t , x);
> title ( 'Título del gráfico');
> xlabel ( 'Magnitud eje X y [unidades]');
> ylabel ( 'Magnitud eje Y y [unidades]');
> legend( 'Descripción de la curva')
```

Graficando varias curvas juntas

Generalmente es necesario graficar distintas curvas superpuestas para compararlas. Hay muchas formas de hacer esto.

Esta es la más elegante:

```
> plot (t1, x1, t2, x2, t3, x3); % etc
```

Y podemos decorar cada curva de la forma:

```
> plot (t1, x1, 'color', 'r', 'linewidth', 3, ...
        t1, x1, 'color', 'b', 'linestyle', '--', ...
        t3, x3, 'color', 'k', 'linestyle', ':' ); % etc
```

Nota: Con los tres puntos le decimos a Octave que el comando sigue en la próxima linea.

Graficando varias curvas juntas

Otra forma, un poco más ~~enferma~~ rebuscada es pasar todos los datos de las absisas en una matriz, como columas o filas según sean compatibles las dimensiones.

Por ejemplo:

```
> t = linspace(0, 10, 100)';
> x = [t, t.^2/10, t.^3/100, t.^4/1000];
> plot (t, x);
> legend ("Cuadrática", "Cúbica", "Cuarta");
> xlabel ("t");
> ylabel ("t^n / 10^n");
```

Esto puede ser útil cuando la matriz x la generamos automágicamente.

Graficando varias curvas juntas

La forma que se usa el 99% de las veces es con el comando **hold**.

Con **hold on** le decimos a Octave que grafique una curva sobre la otra, sin borrar la anterior.

```
> t = linspace(0, 10, 100)';
> x = t.^2;
> td = (0:10)';
> xd = td.^2 + 2*randn(length(td), 1); % ruido
> hold on;
> plot (t, x, 'b');
> plot (td, xd, 'or', 'markersize', 7);
> legend ('Modelo', 'Mediciones');
```

Usos y costumbres al plotear

El “estándar de facto” a la hora de hacer curvas es el siguiente:

```
> close all; % cierro todos los plots
> % bla bla bla
> figure; % creo una nueva figura, abre una ventana
> hold on;
> plot (...); % realizo el primer plot
> plot (...); % realizo los siguientes
> legend (...);
> xlabel (...); ylabel (...);
> print ('-dpng' 'miplot'); % Guardo el plot
```

Ejes (semi)logarítmicos

Si es necesario usar una escala logarítmica para alguno de los ejes (o ambos), es exactamente igual que antes, sólo que hay que usar `semilogx()`, `semilogy()` o `loglog()` en lugar de `plot()`.

Bug al usar ejes logarítmicos y hold

En caso de querer usar `hold on` junto con ejes logarítmicos (o semi), primero es necesario plotear la primer curva y *después* usar el comando `hold on`. Caso contrario, `hold on` crea ejes lineales.

Es decir:

```
> figure; % creo una nueva figura, abre una ventana
> semilogx (...); % primer plot
> hold on;
> semilogx (...); % realizo los siguientes
```

Ejes dobles independientes

Si queremos graficar dos curvas con ejes independientes, podemos usar el comando **`plotyy()`**. La sintaxis es:

```
plotyy (x1, y1, x2, y2);
plotyy (x1, y1, x2, y2, @TipoPlot1, @TipoPlot2);
```

En TipoPlot debemos poner alguna de las funciones de ploteo que ya conocimos (u otras). Esto es un ”puntero a función”. Ya lo veremos luego ;) pero básicamente le decimos a Octave con qué función debe graficar cada curva con sus respectivos ejes.

Ejes dobles independientes

Este ejemplo de la documentación de GNU Octave aclara los tantos:

```
> x = 0:0.1:2*pi;
> y1 = sin (x);
> y2 = exp (x - 1);
> ax = plotyy (x, y1, x - 1, y2, @plot, @semilogy);
> xlabel ("X");
> ylabel (ax(1), "Axis 1"); % Especifico el eje a rotula
> ylabel (ax(2), "Axis 2");
```

Más funciones de ploteo

Hay otras funciones, por ejemplo, para hacer histogramas (hist), gráficos discretos (stem), paretos (?), graficos polares, etc.

Para más info, escribir: `doc plot` en consola.

Gráficos Tridimensonales

Graficos en 3D

Em... no me emociona escribir sobre esto ¿quién usa estas cosas? :P

Ya con lo que vimos, debe ser trivial leer lo de `doc plot3`.

Si no, este enlace: `https://www.gnu.org/software/octave/doc/interpreter/Three_002dDimensional-Plots.html`

Tarea integradora

Tarea propuesta

Supongamos que queremos ”emular”[7] un TP de Físca I. Tenemos un sistema físico, real (un péndulo, un resorte, etc), sobre el cual realizamos mediciones (con cierta incerteza) y un modelo matemático (como la ley de Hooke).

En base a las mediciones obtenidas podemos caracterizar el sistema que medimos (en español: obtener el valor de k del resorte).

[7] Eufemismo para dibujar.

Tarea propuesta

Proponemos:

- Elegir un sistema físico (resorte, péndulo, resonador cuántico, etc).
- Dado algún modelo matemático, trazar curvas de alguna magnitud medible (estiramiento vs fuerza, por ejemplo).
- Simular que realizamos mediciones para levantar dicha curva. Esto es: Dados puntos determinados en las absisas, obtener las ordenadas. Para simular la incerteza de medición, añadir ruido gaussiano con `randn()` o uniforme con `rand()`.
- Ajustar por mínimos cuadrados para obtener los parámetros del sistema (ejemplo, la k) dadas las mediciones con incerteza. Usar `fit()` ~~(o hacer lo de la pseudoinversa de Moore-Penrose)~~.
- Comparar la curva ideal con la curva ajustada.