GNU Octave

Curso rápido - Parte 1 (25 de marzo de 2019)

Fernando Danko, Vero Bustamante

1er cuat. 2019

LABI – FIUBA

Temario

1. LABi Cursos
2. Introducción a GNU Octave
3. Octave como calculadora
4. Trabajando con matrices
5. Gráficos Bi-dimensonales

LABi Cursos

LABi

¿Qué se puede hacer en el labi?

- Estudiar
- Hacer TPs
- Pedir herramientas
- Usar herramientas del pañol
- Pedir libros de la biblioteca especializada
- Tomar mate/comer

Dejando siempre las mesas limpias :)

Cursos del LABi

- Soldadura (Intro, técnica digital, labo de micros, circuitos)
- Software libre:
  - LaTeX(Todas las materias donde haya que presentar informes, papers, tesis)
  - Octave (Análisis de circuitos, procesos estocásticos, señales, redes neuronales)
  - KiCad (Labo de micros, circuitos)
  - Introducción a la consola
  - Modelado 3D
  - Python (énfasis en la utilización de las bibliotecas científicas)
- Software gratuito:
  - LTSpice (Análisis de circuitos, circuitos)
- Y otros...

Introducción a GNU Octave

GNU Octave

GNU Octave es lenguaje de programación de alto nivel destinado principalmente a *cálculo numérico*. Posee capacidades de *resolución de sistemas lineales y no lineales*, y para la realización de *simulaciones* numéricas. Proporciona extensas *capacidades gráficas* para visualización y manipulación de datos.[1]

GNU Octave es *software libre*.

Es *multiplataforma*: corre sobre sistemas GNU/Linux (incluido Android), BSD, Windows y OS X.

[1] Tomado de `https://www.gnu.org/software/octave/`

Interfaz gráfica

Octave (desde la versión 4.0 en adelante) posee una interfaz gráfica por defecto. Vamos a trabajar sobre ella.

Si tenés una versión anterior de Octave (a partir de la 3.8), podés iniciar la interfaz gráfica así:

En UNIX[2] Ejecutar en consola `$ octave --force-gui`

En Windows Botón derecho en el acceso directo > Propiedades. En el campo destino, agregar al final `--force-gui` y apretar Aceptar.

[2] Linux, OS X, BSD, etc.

Interfaz gráfica - Paneles

La interfaz de Octave está compuesta de varios paneles:

- Command Window: Aquí se ejecutan los comandos y observan los resultados
- Editor: Un editor de archivos de texto incorporado
- File Browser: Muestra el contenido de la carpeta de trabajo
- Workspace: Muestra las variables definidas
- Command History: Muestra los comandos utilizados anteriormente
- Current Director: Muestra el directorio de trabajo. (MUY IMPORTANTE)

Interfaz gráfica - Personalización

La interfaz de Octave está compuesta de varias ventanitas separables (paneles). Se pueden agrupar como pestañas si tira una sobre otra con el mouse, o ocupando un espacio fijo si se la suelta sobre un lugar disponible.

A continuación algunos ejemplos:

Interfaz gráfica - Personalización

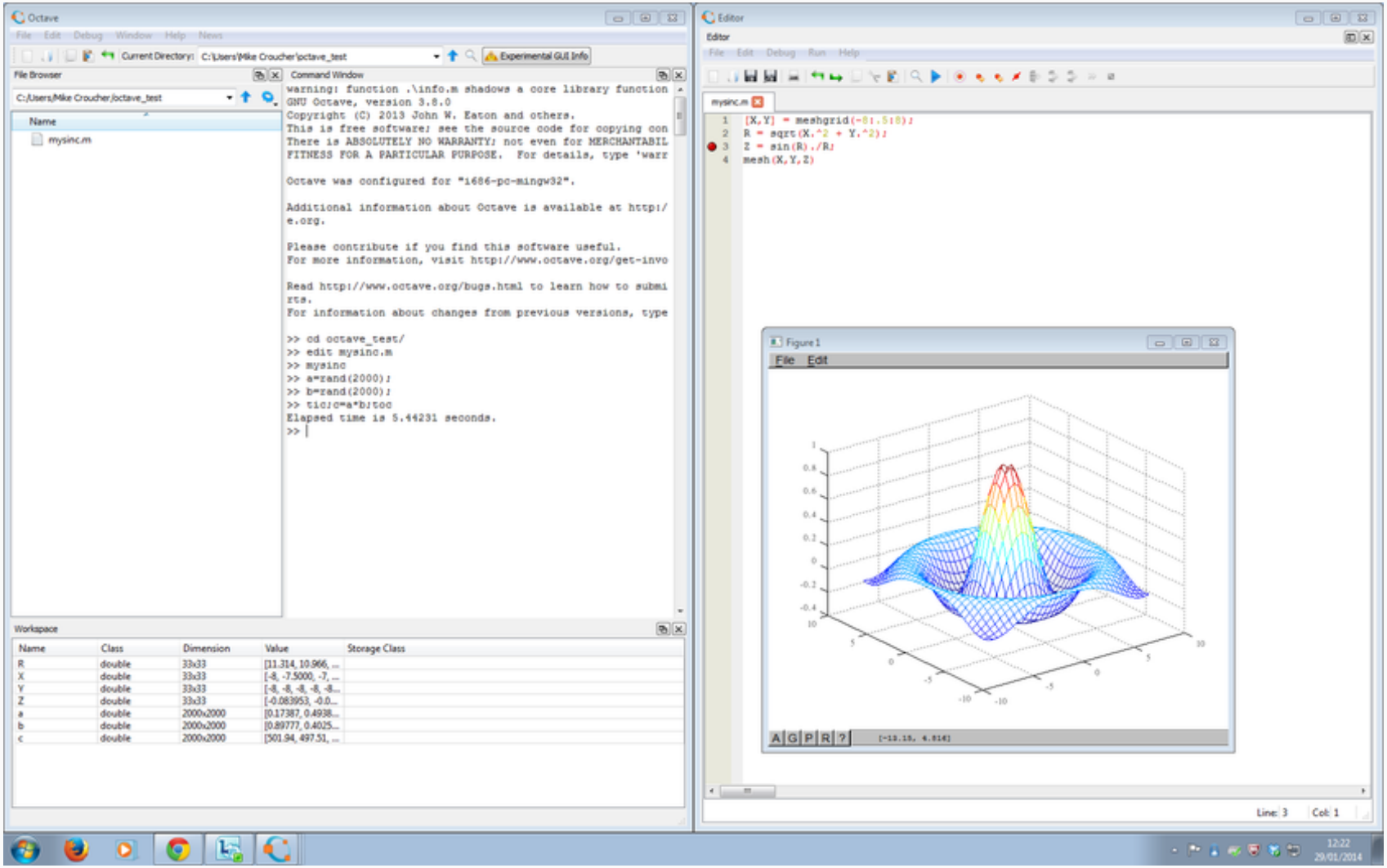

Figura 1: Editor en ventana aparte

Interfaz gráfica - Personalización

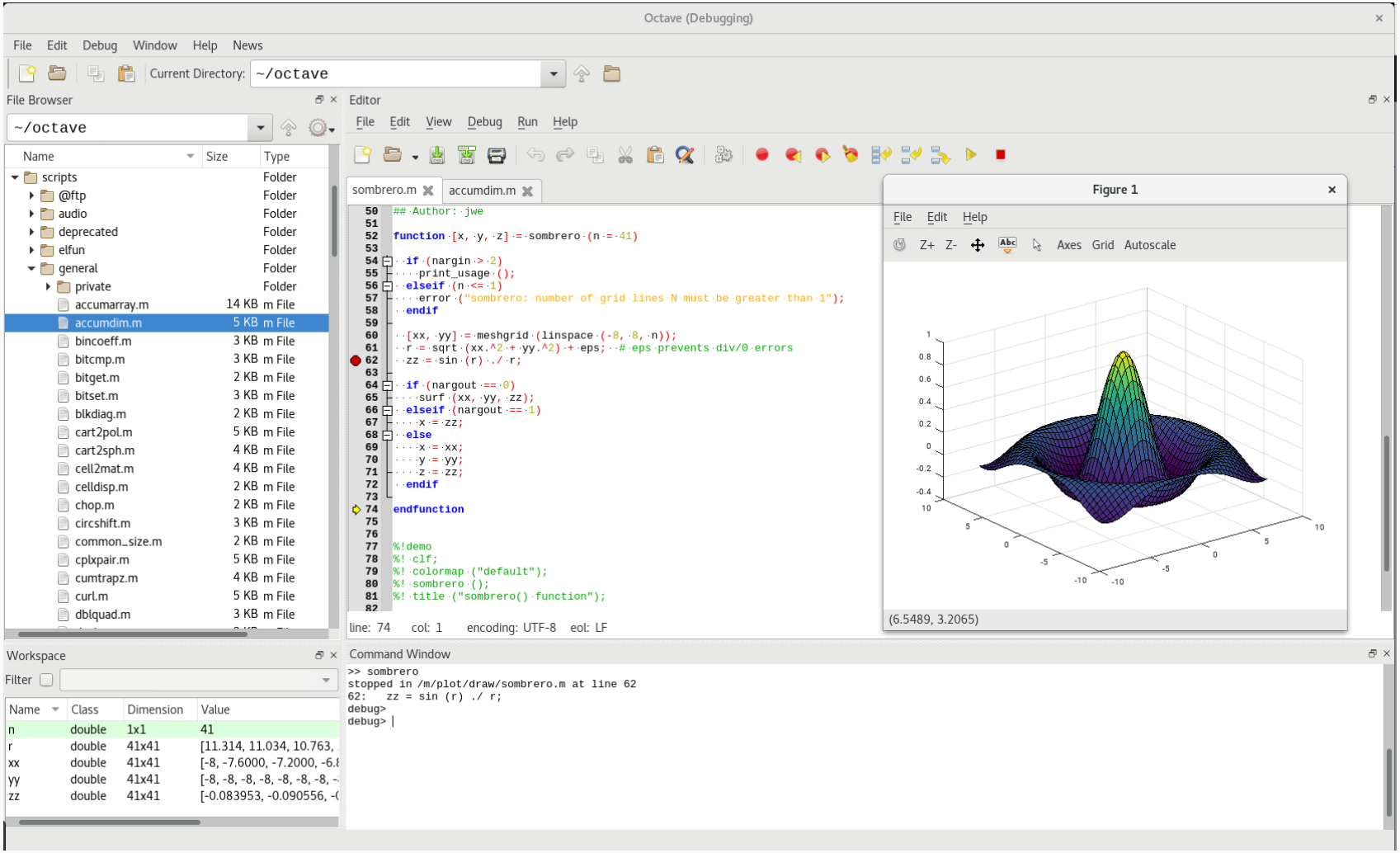

Figura 2: Consola en parte inferior

Interfaz gráfica - Personalización

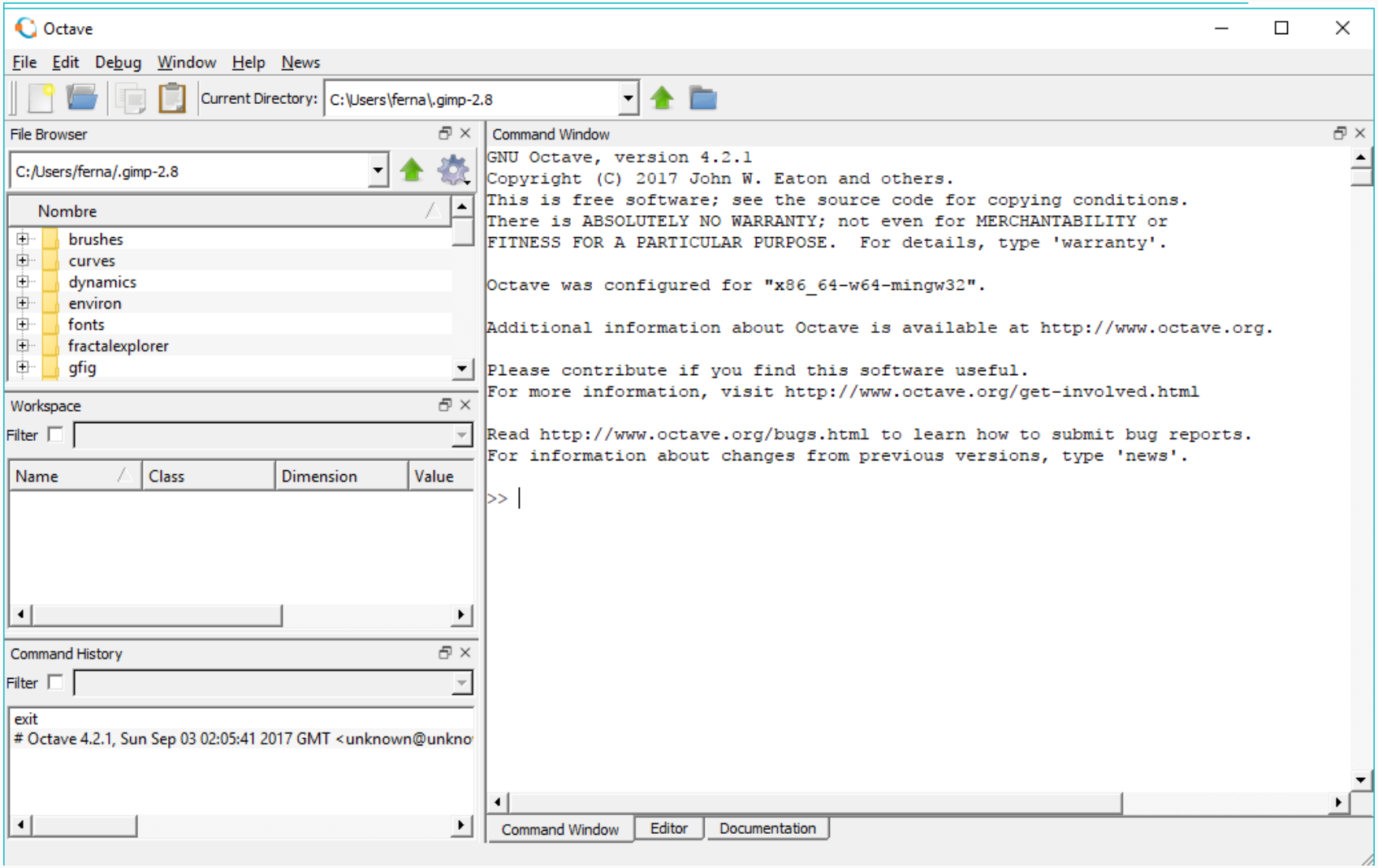

Figura 3: Formato por defecto (feo)

Importancia de la ventana de Comandos

Octave se comunica con nosotros a través de la ventana de comandos.

Octave se tildó :(

Muchas veces sucede que nuestro código no ejecuta, o parece que se queda tildado el programa. Lo primero que debemos hacer es abrir el Panel *Command Window* y leer lo que diga allí.

Importancia de la ventana de Comandos

Casi todas las operaciones que hacemos a través de la interfaz gráfica de Octave pueden realizarse por medio de comandos.

(Por ejemplo, al tocar el botón Run, el programa escribe un comando por nosotros)

Octave como calculadora

Calculadora simple

Octave puede usarse como calculadora. Simplemente podemos escribir operaciones matemáticas en *Command Window* (de aquí en más, la Consola) 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[3]. 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 :)

[3] 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*[4] *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
```

[4] 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 = [% tiempo (seg), posición (m)
              0.0      1.30
              1.0      6.40
              1.71    16.1
              2.43    31.1
              3.14    51.0
              ];
```

Se pueden recuperar las filas o columnas de esta forma:

```
tiempo = mediciones(:,1); % fila: todas, columna: 1
posicion = mediciones(:,2); % fila: todas, columna: 2
inicial = mediciones(1,:); % fila: 1, columna: todas
```

Truquitos

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[5]

¡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.

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

Gráficos Bi-dimensonales

Ejemplo de grafico de dispersión

Retomando los datos anteriores:

```
t = [0, 1.0, 1.71, 2.43, 3.14];
y = [1.3, 6.4, 16.1, 31.1, 51.0];
```

Se puede plotear la posición en función del tiempo con el comando:

```
plot(t, y);
```

Ejemplo de grafico de dispersión

Se debe obtener algo así:

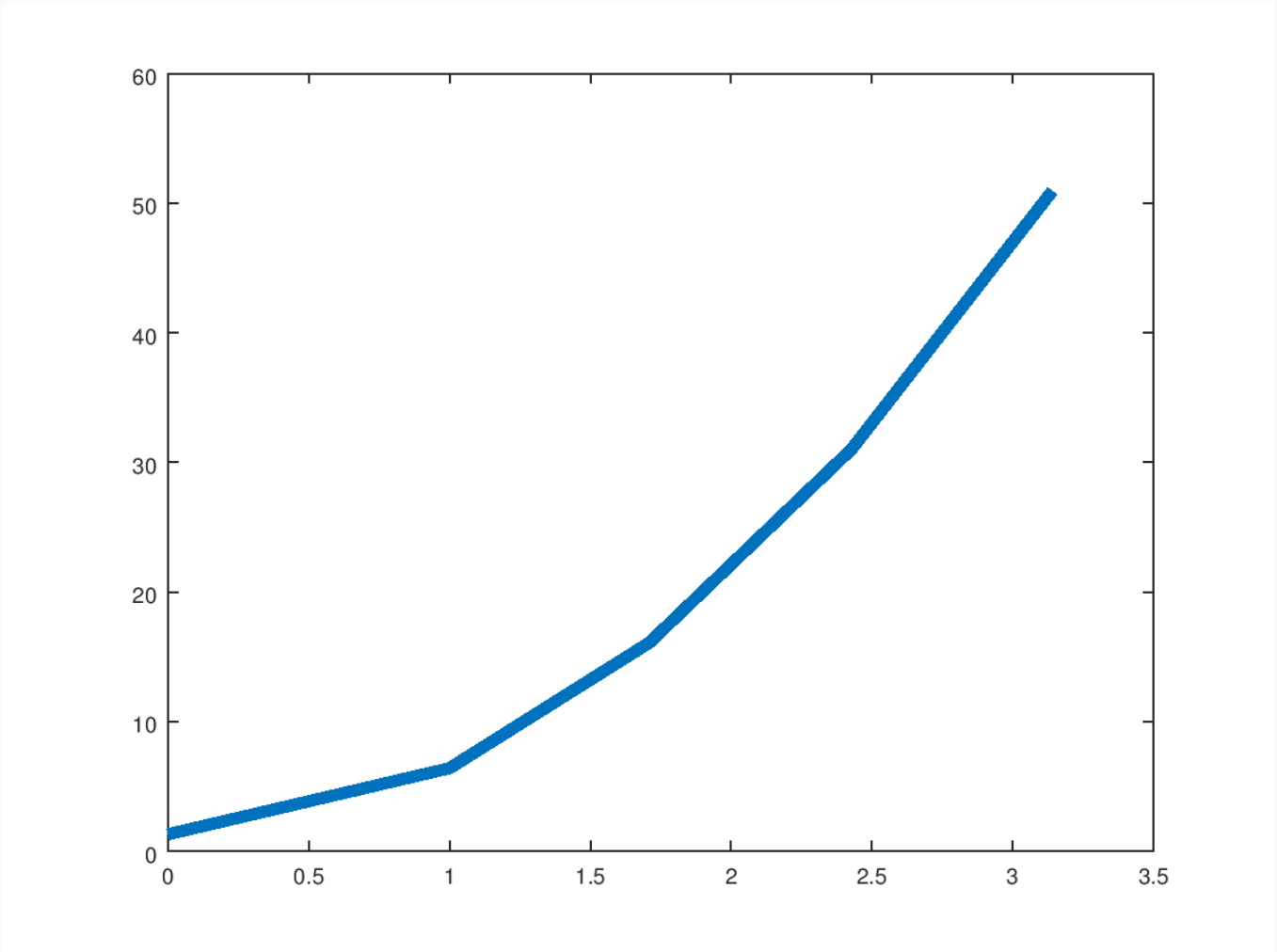

Figura 4: Una parabola media quebrada

Ejemplo de grafico de dispersión

Recordemos que Octave está ploteando una serie de puntos en el plano (x,y). Por defecto, une estos puntos con rectas, pero podemos especificar otro formato.

```
plot(t, y, 'o'); % circulos
plot(t, y, 'rx'); % en rojo, cruces
plot(t, y, 'rx:'); % en rojo, cruces, linea puntada
```

Para ver todos los formatos disponibles ejecutar `help plot`

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()

Hay formas más humanas de escribir el formato:

```
plot (t , y , 'color' , 'r' , 'linestyle', ':')
```

Aquí 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 (pareto), graficos polares, etc.

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

Datos en tabla

Una forma conveniente de presentar los puntos a plotear es mediante una tabla, por ejemplo:

```
mediciones = [% tiempo (seg), posición (m)
              0.0      1.30
              1.0      6.40
              1.71    16.1
              2.43    31.1
              3.14    51.0
              ];
```

Aquí se puede plotear de la siguiente forma:

```
plot(mediciones(:,1), mediciones(:,2));
```