

# Gráficos en Matlab

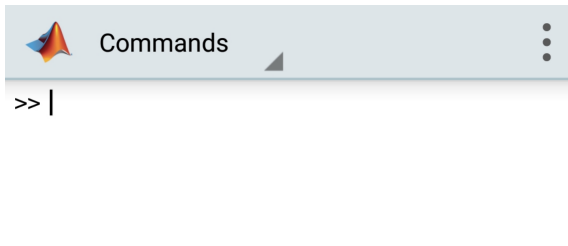
14 de mayo de 2019

# Matlab

- ▶ Software especializado en Análisis Numérico.
- ▶ Muy usado en la industria y la academia.
- ▶ Es pago.
- ▶ Hay versión móvil: MATLAB Mobile. Es gratuita (hay que registrarse).
- ▶ Para la compu, existe un clon **gratuito**: Octave.

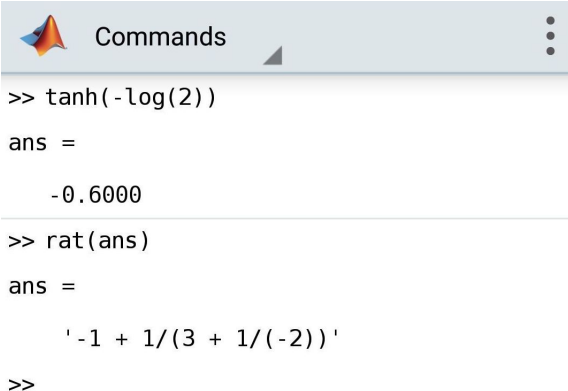
## MATLAB mobile

- ▶ Está disponible para Android y iPhone.
- ▶ Opera desde la nube (¡consume datos!)
- ▶ Luce así:



## Comandos básicos:

La consola de Matlab opera como una super-calculadora. Así, se pueden hacer operaciones complejas:



```
Commands
>> tanh(-log(2))
ans =
    -0.6000

>> rat(ans)
ans =
    '-1 + 1/(3 + 1/(-2))'

>>
```

Da los resultados en *punto flotante*, **no** devuelve fracciones. El comando `rat` devuelve la fracción más cercana al número.

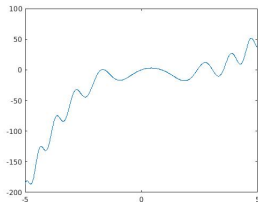
## Gráficos de funciones

Para graficar funciones hay que definir una variable (como si fuera un vector) y su imagen, y usar el comando `plot`.

Por ejemplo, para graficar

$$f(x) = x^3 - 3x^2 + 14 \sin(x(1-x))$$

```
>> x = linspace(-5,5,1000);  
>> y = x.^3-3*x.^2+14*sin(x.*(1-x));  
>> plot(x,y)
```



`linspace(-5,5,1000)` crea un vector que comienza en  $-5$ , termina en  $5$  y tiene 1000 casilleros equiespaciados.

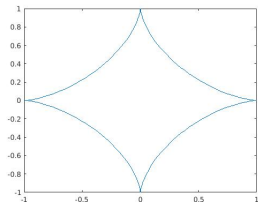
El `;` evita que se muestre el resultado (no queremos ver los 1000 casilleros).

Para las operaciones que implican *productos* entre vectores hay que poner un punto: `.*`, `.^` y `./`.

## Curvas paramétricas

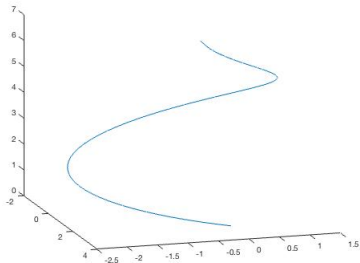
Lo mismo vale para graficar usando parametrizaciones de curvas:

```
>> t = linspace(0,2*pi,200);  
>> x = cos(t).^3;  
>> y = sin(t).^3;  
>> plot(x,y)
```



En  $\mathbb{R}^3$ , el comando es `plot3`:

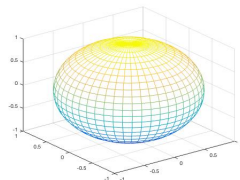
```
>> t = linspace(0,2*pi,200);  
>> x = sqrt(t).*cos(t);  
>> y = sqrt(t).*sin(t);  
>> z = 2*pi-t;  
>> plot3(x,y,z)
```



## Superficies

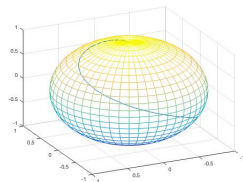
Para superficies hay que definir el dominio de la parametrización con `meshgrid` y graficar con `mesh`:

```
>> theta = linspace(0,2*pi,50);  
>> phi = linspace(0,pi,25);  
>> [u,v] = meshgrid(theta,phi);  
>> x = sin(v).*cos(u);  
>> y = sin(v).*sin(u);  
>> z = cos(v);  
>> mesh(x,y,z)
```



Se pueden hacer dos dibujos juntos. Por ejemplo, para graficar sobre la esfera la curva dada por  $\theta = 2\phi$ , debo agregar:

```
>> hold on  
>> w = [0:0.1:pi];  
>> xc = sin(w).*cos(2*w);  
>> yc = sin(w).*sin(2*w);  
>> zc = cos(w);  
>> plot3(xc,yc,zc)
```



`hold on` retiene el gráfico anterior para seguir graficando sobre él.

`[0:0.1:pi]` genera un vector que va de 0 a  $\pi$  con paso 0,1.

## Superficies de revolución

Para graficar superficies de revolución hay dos alternativas:

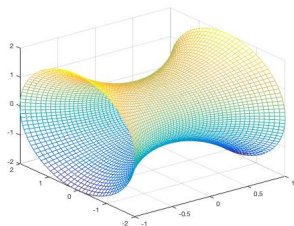
### 1) Calcular la parametrización a mano y graficarla como antes.

Supongamos que queremos la superficie de revolución de la función  $y = x^2 + 1$  en torno al eje  $x$ , para  $x \in [-1, 1]$ .

La rotación generará círculos en torno al eje  $x$ , por lo que tenemos que hacer polares en las variables  $(y, z)$ , obteniendo algo de la pinta:  $(x, r \cos(t), r \sin(t))$ . A su vez, para un  $x$  dado, el radio del círculo es  $r = y = x^2 + 1$ , por lo cual la parametrización queda:

$$\phi(t, x) = (x, (x^2 + 1) \cos(t), (x^2 + 1) \sin(t))$$

```
>> X = [-1:0.01:1];  
>> theta = linspace(0,2*pi,100);  
>> [x,t] = meshgrid(X,theta);  
>> y = (x.^2+1).*cos(t);  
>> z = (x.^2+1).*sin(t);  
>> mesh(x,y,z)
```





## Superficies de revolución

### 2) Aprovechar el comando `cylinder`.

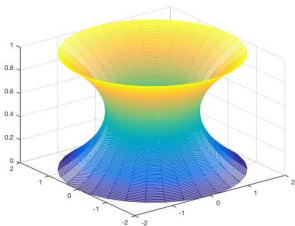
`cylinder` toma dos variables:  $r$  y  $n$  y genera las variables correspondientes a un cilindro (de radio variable) **en torno al eje  $z$** .

En nuestro ejemplo, tenemos que cambiar las variables para que el eje de rotación sea  $z$ . Por lo tanto, graficaremos la revolución de  $y = z^2 + 1$  en torno a  $z$ .

El parámetro  $r$  es el vector de radios del cilindro (calculados desde  $z$ ). Como antes,  $r = y = z^2 + 1$ .

El parámetro  $n$  es el número de particiones que se harán en el ángulo  $\theta$ . El código queda así:

```
>> Z = [-1:0.01:1];  
>> r = Z.^2+1;  
>> n = 30;  
>> [x,y,z] = cylinder(r,n);  
>> mesh(x,y,z)
```



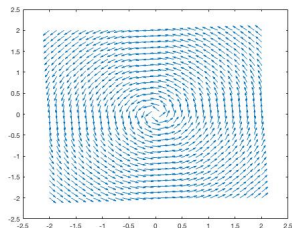
Para obtener el gráfico deseado, basta con rotar el gráfico para que el eje  $z$  se vea donde dibujaríamos el eje  $x$ .

**OJO:** Recordar que  $r$  siempre se mide desde el eje  $z$ . ¿Quién sería  $r$  si se quiere la rotación de la curva  $z = x^2$ , en torno a  $z$ ?

## Campos vectoriales

Por último para graficar campos vectoriales, es necesario generar cuatro variables:  $x$  e  $y$  del dominio y  $u$  y  $v$  representando las componentes del campo.

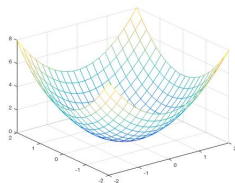
```
>> [x,y] = meshgrid(linspace(-2,2,200));  
>> u = -y./sqrt(x.^2+y.^2);  
>> v = x./sqrt(x.^2+y.^2);  
>> quiver(x,y,u,v)
```



## Yapa

Puede resultar útil comparar el gráfico de una función  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  (superficie) y de su gradiente (campo vectorial). Por ejemplo, consideremos  $f(x, y) = x^2 + y^2$

```
>> [x,y] = meshgrid(linspace(-2,2,40));  
>> z = x.^2+y.^2;  
>> mesh(x,y,z)
```



El comando `contour` grafica curvas de nivel de la superficie. Podemos graficar, por ejemplo:

```
>> [x,y] = meshgrid(linspace(-2,2,40));  
>> z = x.^2+y.^2;  
>> contour(x,y,z)  
>> [fx,fy] = gradient(z);  
>> hold on  
>> quiver(x,y,fx,fy)
```

