GNU Octave

Curso rápido - Parte 2 (1 de abril de 2019)

Fernando Danko, Vero Bustamante

1er cuat. 2019

LABI – FIUBA

Temario

1. Segunda Clase

2. Índices y Rangos

3. Graficos de superficies

4. Funciones

Segunda Clase

Índices y Rangos

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  % no tiene paso, se toma como 1
1  2  3  4  5
> 1:2:10  % se saltea de a 2
1  3  5  7  9
> 15:-1:10  % paso negativo, disminuye en 1 hasta 10
15  14  13  12  11  10
```

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
```

Revirtiendo 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];
> v_rev = v(end:-1:1) %end indica el final del vector
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).

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` 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.

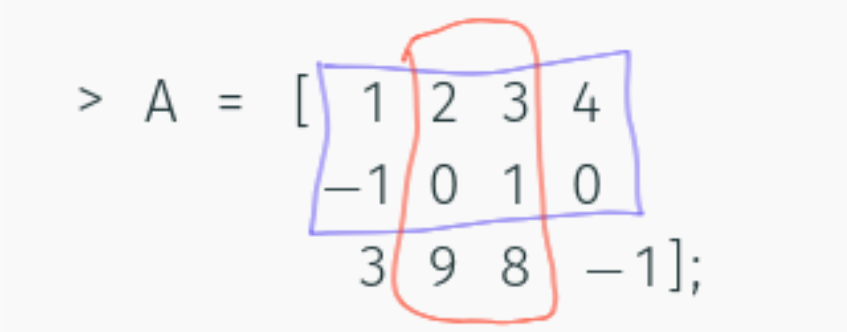
Índices en matrices - submatrices

Ejemplo, sea:

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

Podemos jugar con lo que aprendimos de rangos:

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



Esto indica que obtiene las filas de 1 a 2 y las columnas de 2 a 3.

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) % primer fila y última columna
4
> A(end-2, end-1) % antepenúltima f. y anteúltima c.
3
> A(:, end-1) % todas las filas y anteultima columna
3
1
8
```

Seguimos indexando

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
```

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], :)   % Selecciono las columnas
   1   2   3   4
   1   2   3   4
  −1   0   1   0
   1   2   3   4
```

Técnicas avanzadas de indexado[1]

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>=4)
8.1  9.0  9.1  6.3

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

%puedo aplicarle una función por fuera
%para saber cuántos cumplen
> length (v (v >=4))
ans =  4
```

[1]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)

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

Nota: ¿ese ”dado qué”no les suena de algún lado? ¿Para qué suponen que se usa algo así?

Técnicas avanzadas de indexado: operadores lógicos

`&` Operador “y” lógico (*and*)

`|` Operador “o” lógico (*or*)

`~` Operador “no” lógico (*not*)[2]

```
> 1 | 0
ans = 1
> 1 & 0
ans = 0
> ~1
ans = 0 % variable tipo lógica
> t=1:2:9
    1     3     5     7     9
> t(t>3 & t<9)
    5     7
```

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

Técnicas avanzadas de indexado

¿Y si quiero saber qué índices de un vector **v** son los que cumplen que $a < v < b$?

Para eso existe el comando `find()`.

```
> notas = 2:1:10;
 2 3 4 5 6 7 8 9 10

> indices = find ( notas>=4 & notas<=7)
3   4   5   6
```

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

> x(   t>2.2 & t<3 )         % Sí
```

< Mencionar usos de los índices en la práctica >

Graficos de superficies

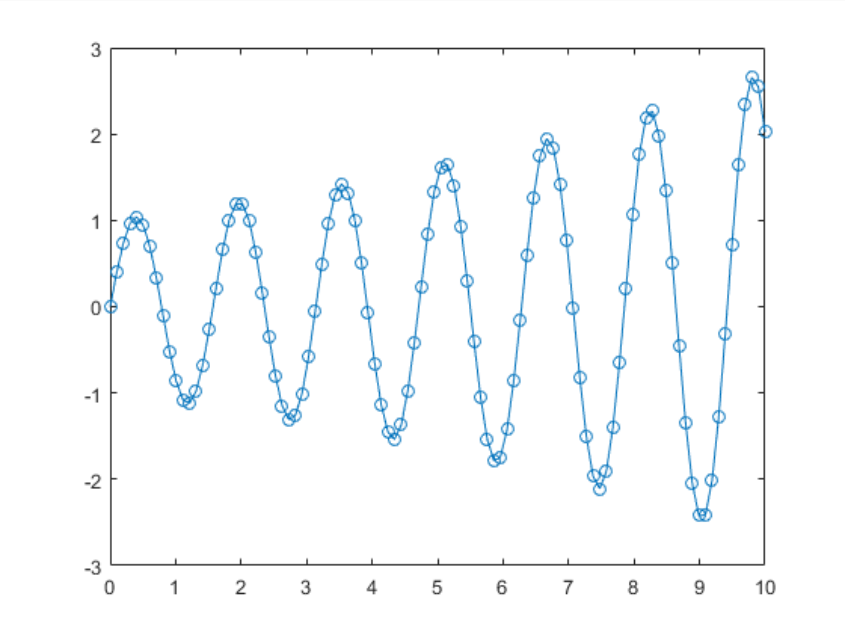
Superficies

A la hora de graficar una función de dos variables, necesitamos posicionarnos en un espacio tridimensional.

El dominio lo tomamos sobre el plano xy, y la grafica de la función sobre el eje z.

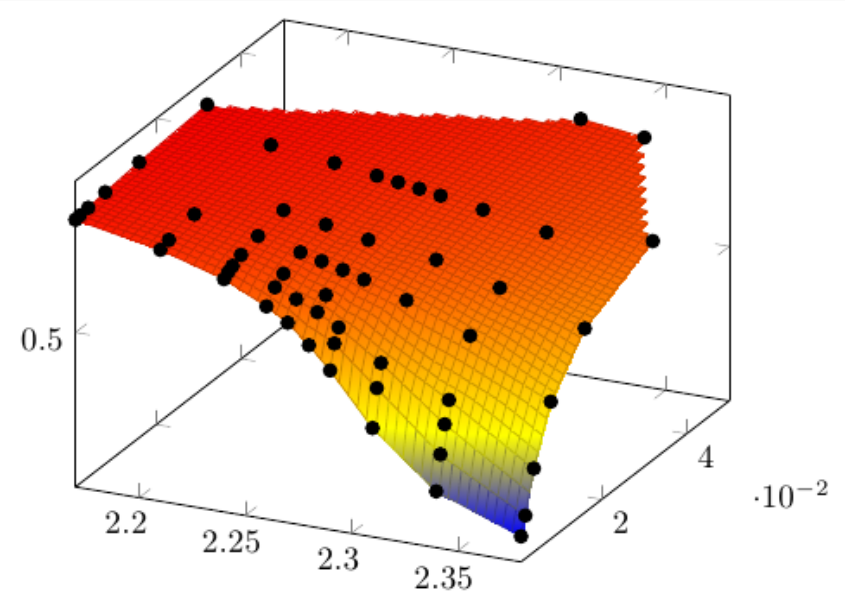
Superficies

Para un gráfico de una función, necesitamos discretizar el dominio, generalmente con linspace. Octave se encarga de llenar con lineas los valores intermedios para representar el plot.



Superficies

En el caso de un plot en dos dimensiones, necesitamos una cuadrícula (malla o mesh), sobre la cual evaluar la función:



Superficies

Para crear el dominio de nuestro campo escalar, debemos hacer el producto cartesiano entre los valores de x e y. Para ello, usamos meshgrid:

```
x = linspace(-1,1,100);
y = linspace(-2,2,500);

[xx,yy] = meshgrid(x,y);

surf(xx,yy, (xx.^2)+(yy.^2)/2)
```

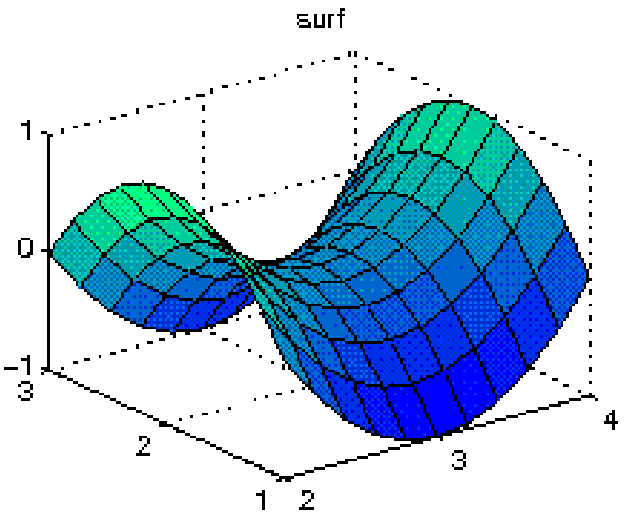
Superficies

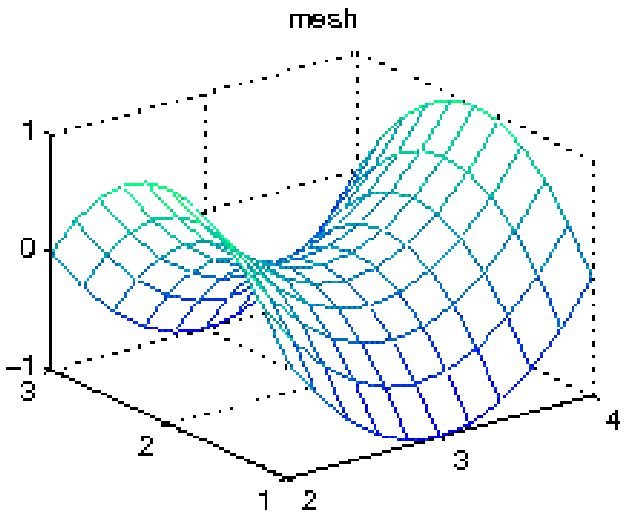
Cabe destacar que los xx e yy que devuelve meshgrid son matrices (por comodidad al representarlos).

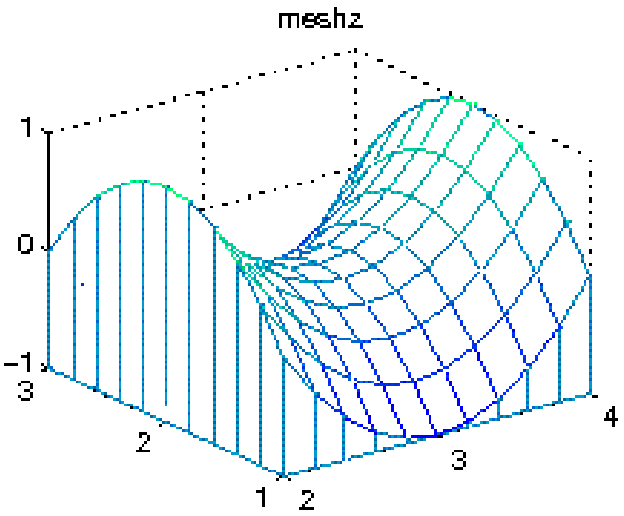
Sin embargo, muchas funciones no se portan bien al recibir como parametro una matriz, por lo cual, podemos vectorizar el resultado:

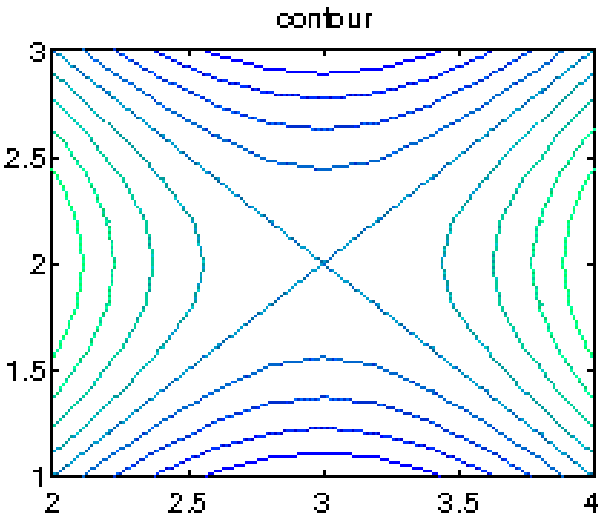
```
x = linspace(-1,1,100);
y = linspace(-5,5,500);

[xx,yy] = meshgrid(x,y);

xxv = xx(:);
yyv = yy(:);

surf(xxv,yyv, alguna_f(xxv,yyv))
```

Hay varias funciones para plotear

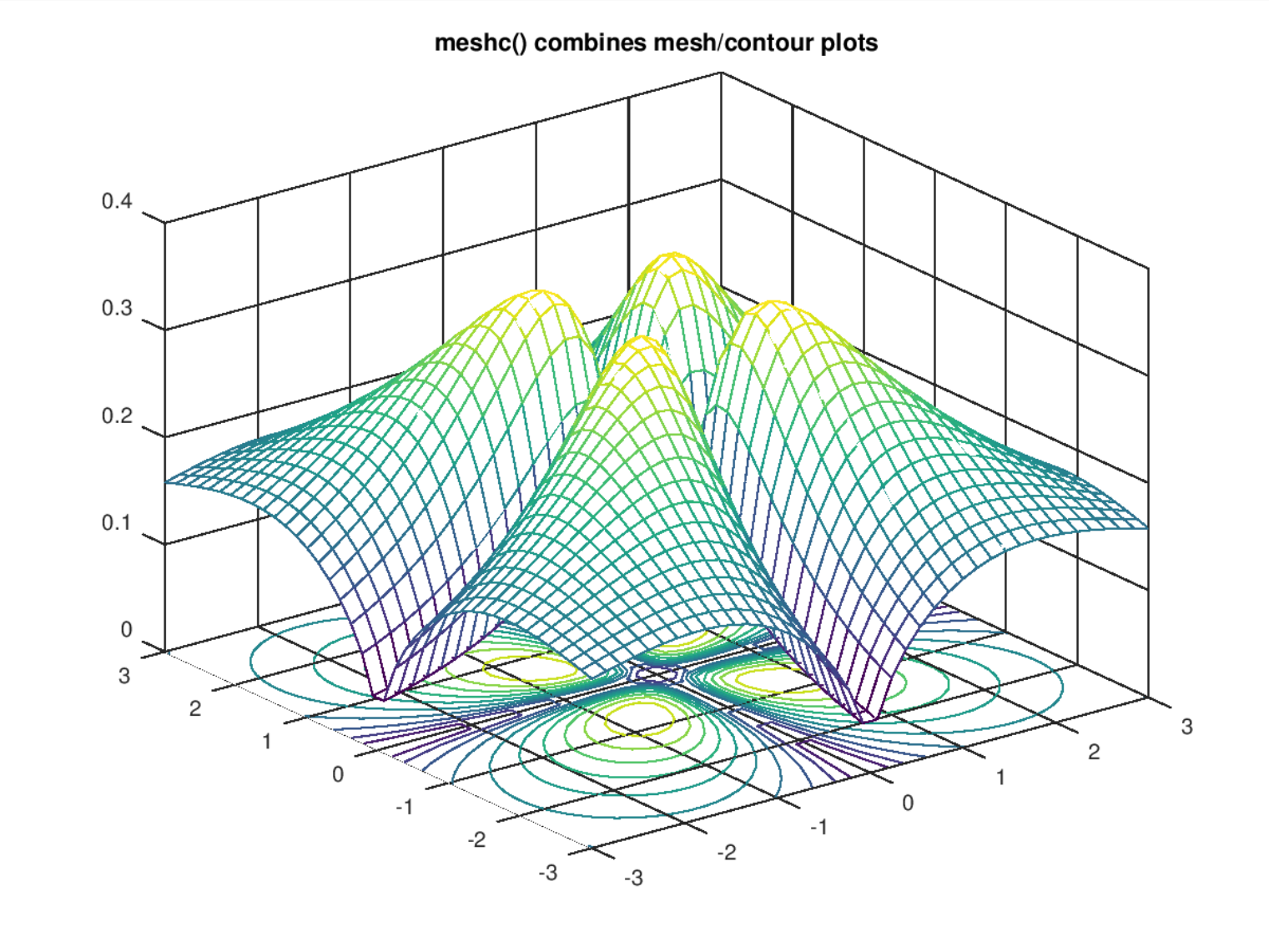




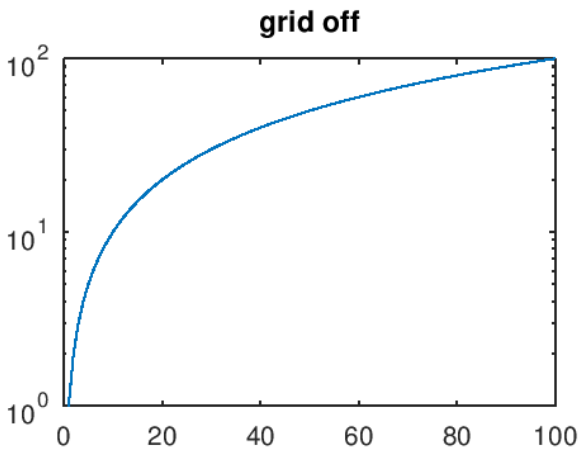


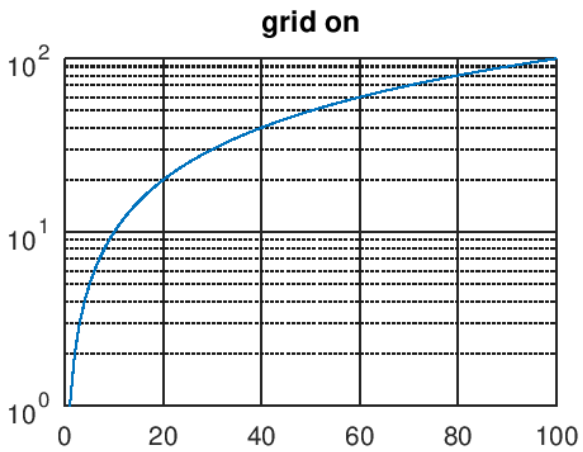


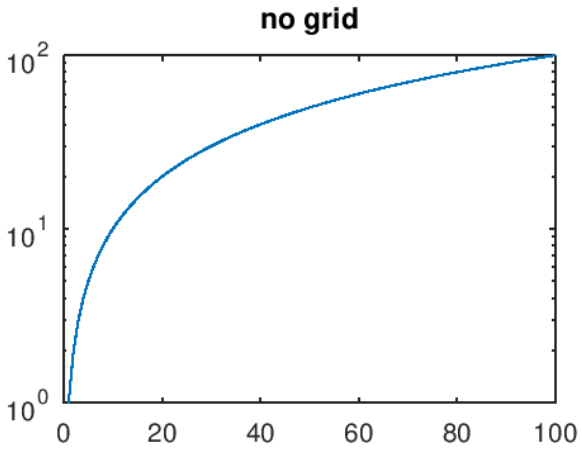
Hay varias funciones para plotear

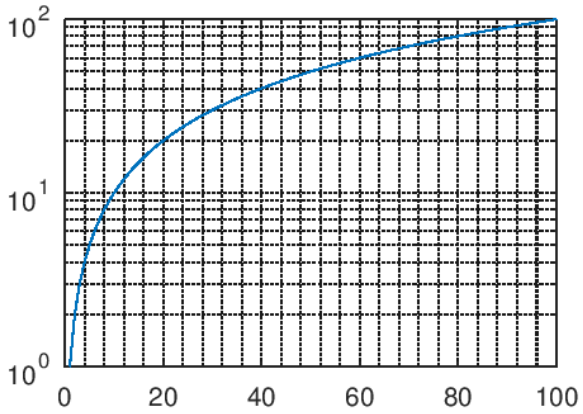


Notas sobre grid

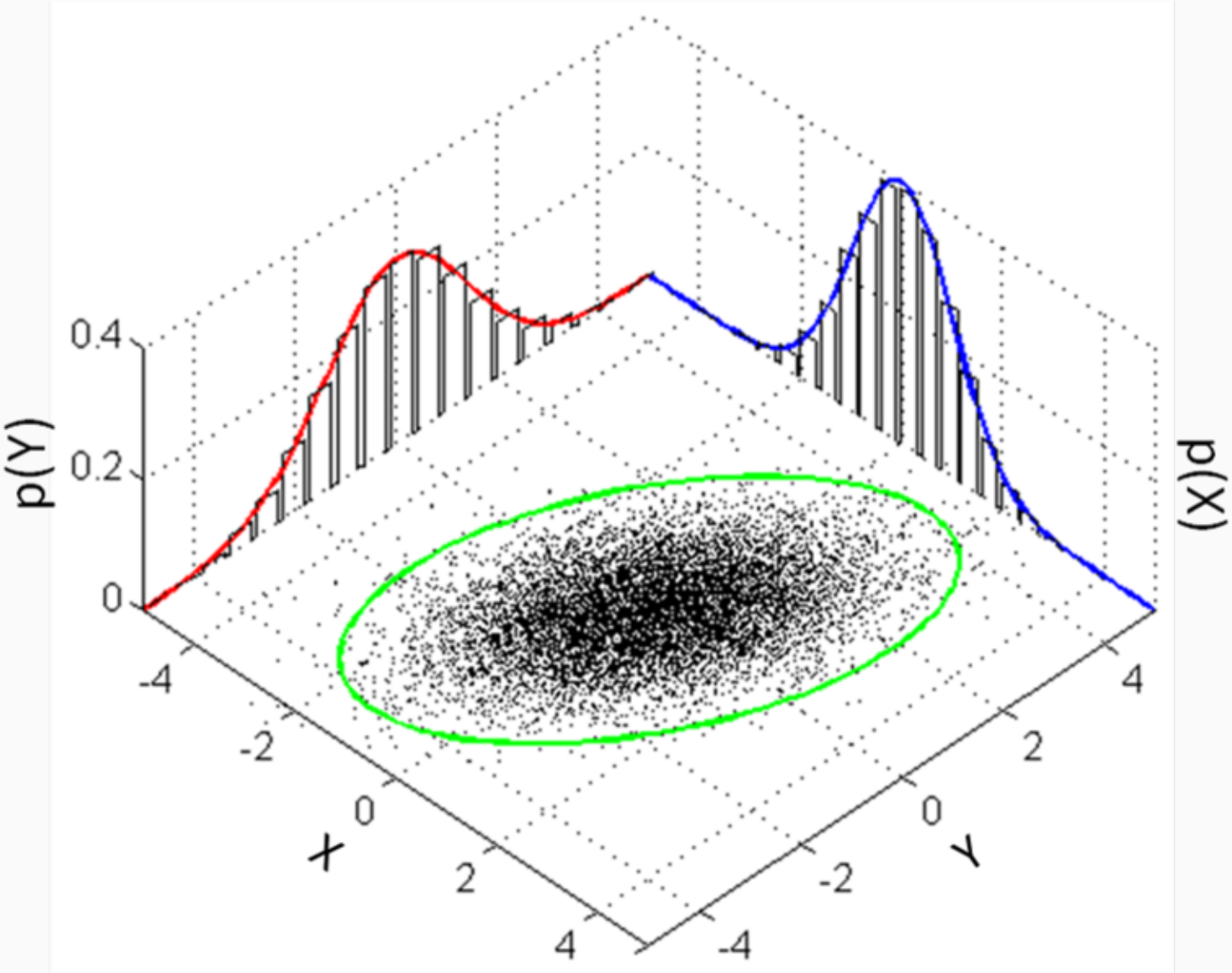








Tarea: Mezcla de normales bivariadas



Funciones

Generalidades

- Funciones internas de Octave
- Funciones anónimas
- Funciones en archivos separados
- Funciones secundarias

Generalidades

Las funciones pueden tomar ninguno, uno o varios datos, los procesan, y devuelven el resultado.

Función Max

```
> v=[1:1:5  7:-2:1]
v =    1    2    3    4    5    7    5    3    1
> max(v)
ans =   7
> v=[1  2  500j]
v =   1 +    0i       2 +    0i    0 + 500i

> max(v)
ans =      0 + 500i  % la de mayor modulo
```

La función ”min”funciona igual.

Función Mean y funciones round, ceil y floor

```
> A =    [1    2
          3    4
          1    2
          1    5
          1    7];
> mean(A)
    1.4000    4.0000
> mean(A,1)
    1.4000    4.0000

> u= mean(A,2)
    1.5000
    3.5000
    1.5000
    3.0000
    4.0000
```

Función Mean y funciones round, ceil y floor

```
> u= mean(A,2)
    1.5000
    3.5000
    1.5000
    3.0000
    4.0000

> ceil(u)'
    2    4    2    3    4
> floor(u)'
    1    3    1    3    4
> round(u)'
    2    4    2    3    4
```

Funciones para matrices

```
>> A=[1  1;1  0];

>> det(A)
ans = -1
>> trace(A)
ans =  1

>> B=[1  1; 1  1];
>> null(B) % nulo de B
   -0.70711
    0.70711
>> orth(A) % BON espacio columna de A
    0.85065    0.52573
    0.52573   -0.85065
```

Funciones para matrices

```
>> [U,S,V]=svd(A)
U =
   -0.85065   -0.52573
   -0.52573    0.85065

S =
Diagonal Matrix

    1.61803         0
          0    0.61803

V =
   -0.85065    0.52573
   -0.52573   -0.85065
```

Funciones para matrices

```
>> [q,r]=qr(A)
q =

   -0.70711   -0.70711
   -0.70711    0.70711

r =

   -1.41421   -0.70711
    0.00000   -0.70711
```

Funciones para matrices

```
>> [S,D]=eig(A)
S =

    0.52573   -0.85065
   -0.85065   -0.52573

D =

Diagonal Matrix

   -0.61803         0
         0    1.61803
```

Imprimir por pantalla

```
>a=12;

>> printf("cantidad de alumnos en el curso: %d \n",a)
cantidad de alumnos en el curso: 12

>> printf("cantidad de alumnos en el curso: %f \n",e)
cantidad de alumnos en el curso: 2.718282
```

Imprimir por pantalla

```
>> printf("cantidad de alumnos en el curso: %.2f \n",e)
cantidad de alumnos en el curso: 2.72

>> printf("cantidad de alumnos en el curso: %.1f \n",e)
cantidad de alumnos en el curso: 2.7

>> printf("cantidad de alumnos en el curso: %.f \n",e)
cantidad de alumnos en el curso: 3

>> printf("cantidad de profesores: %d y %s :% d \n",
a,"alumnos",12)
cantidad de profesores: 10 y alumnos :12
```

Levantar CSV

```
matriz= csvread("archivo.csv");
```

Otros comentarios:

La función `length(v)` permite medir el largo de un vector. Aplicada a una matriz, devuelve la dimensión más grande.

`size(M)` devuelve un vector con las dimensiones de la matriz.

`size(M,1)`: cantidad de filas

`size(M,2)`: cantidad de columnas

`numel(M)`: devuelve la cantidad de elementos de M.

Tarea

Crear un *script* que cuantos estudiantes promocionan de un curso del cbc. El archivo que nos pasan es ”notas.csv”. La condición de promoción es que el promedio del alumno sea mayor o igual a 7 y haya sacado 4 o más en cada parcial.

Forma tradicional de definir funciones

Prototipo para definir una función:

```
function resultado = nombre_funcion (argumentos)
   ... cuentas ...
endfunction % no conviene poner simplemente end
```

Multiples argumentos

Ecuación horaria

```
function posicion = obtener_posicion(t,xo,vo,a)
  posicion = xo + vo*t + 0.5*a*t.^2;
endfunction
```

Aquí simplemente se reciben varios parámetros y se calcula la posición de un MRUV.

Multiples argumentos, multiples salidas

Las funciones de Octave pueden devolver más de un valor:

Ecuación horaria

```
function [x,y] = calcular_tiro(t,xo,yo,vo,angulo,g)
  x = xo + vo*cosd(angulo)*t;
  y = yo + vo*sind(angulo)*t - 0.5*g*t.^2;
endfunction
```

Si hace:

```
>> calcular_tiro(10,0,0,10,35,9.8)
ans =  81.915
```

Solamente se obtiene el primer valor que devuelve la función, esto es, la variable x.

Multiples argumentos, multiples salidas

Ecuación horaria

```
function [x,y] = calcular_tiro(t,xo,yo,vo,angulo,g)
  x = xo + vo*cosd(angulo)*t;
  y = yo + vo*sind(angulo)*t - 0.5*g*t.^2;
endfunction
```

Si hace:

```
>> [posx, posy] = calcular_tiro(10,0,0,10,35,9.8)
posx =  81.915
posy = -432.64
```

Ahora sí, se obtienen los dos valores que devuelve la función.

Condicionales

A veces es necesario verificar si se cumplen o no ciertas condiciones. Para ello contamos con el comando `if`. En el codigo anterior, vamos a devolver y=0 si el valor resultante de y es negativo (cosa que no transpase la tierra)

Ecuación horaria más realista

```
function [x,y] = calcular_tiro(t,xo,yo,vo,angulo,g)
  x = xo + vo*cosd(angulo)*t;
  y = yo + vo*sind(angulo)*t - 0.5*g*t.^2;

  if (y < 0)
    y=0;
  endif
endfunction
```

Condicionales

Ecuación horaria más realista

```
function [x,y] = calcular_tiro(t,xo,yo,vo,angulo,g)
  x = xo + vo*cosd(angulo)*t;
  y = yo + vo*sind(angulo)*t - 0.5*g*t.^2;

  if (y < 0)
    y=0;
  endif
endfunction
```

Ahora:

```
>> [posx, posy] = calcular_tiro(10,0,0,10,35,9.8)
posx =  81.915
posy = 0
```

Iterador for

La estructura básica de un for

```
for i=primer_valor:paso:ultimo_valor
    % acá hacen cuentas
endfor
```

Calculemos un promedio de notas

El promedio de un conjunto de muestras $\{x_i\}_{i=1...N}$ se define como:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{1}$$

```
clc %limpio pantalla
notas = floor(10*rand(1,5)); % notas aleatorias
cantidad=size(notas,2);
total = 0;
for i=1:cantidad
    total+= notas(i);
endfor
promedio = total/cantidad
```

Iterador for

Comentarios del ejemplo anterior:

- Se podría colocar un paso
- La variable `i` se crea para recorrer todos los elementos del vector de datos.
- size(notas,2) equivale a length(notas)

Ejercicio de simulación:

- Armar una función que estime π por Montecarlo. Debe generar N vectores con distribución uniforme [0,1]×[0,1]

$$f_U(x) = \begin{cases} 1 & |x| \leq 1 \\ 0 & |x| > 1 \end{cases} \tag{2}$$

Usar x=rand(N,2) para generarlos todos juntos. Esto me genera pares de vectores en el espacio pedido.

```
> rand(3,2)
    0.231909    0.146372
    0.231932    0.059681
    0.014655    0.389704
```

En este ejemplo tengo 3 pares de vectores.

Ejercicio de simulación:

- Analizar cuáles de estos vectores caen dentro del círculo unitario, usar `norm(x,"rows")<1`.
- Ahora sólo resta contar cuántos vectores cumplen con esto. Usar `nnz(norm(x,"rows")<1)`.
- Finalmente la probabilidad muestral de caer dentro del círculo la calculamos como:
  `P=nnz(norm(x,"rows")<1)/N`
  $P = \frac{\text{Cant muestras de norma < 1}}{\text{Cantidad total de vectores}}$
- Analíticamente, sería: P=Caer en el circulo / P area total,
  $P = (\pi * 1^2/4)/(1) = \pi/4$.
  Por lo cual, estimamos pi=4*P

Ejercicio de simulación:

```
function mypi = estimar_pi (N)
  x=rand(N,2);
  mypi = 4*nnz(norm(x,"rows")<1)/N;
endfunction
```

Ejercicio

Aprobados en Álgebra 2

Construya una función que reciba un archivo por parámetro y devuelva la cantidad de aprobados. El prototipo es el siguiente:

```
function aprobados = aprobar(archivo_notas)
 // cosas
endfunction
```

Luego ejecutar el script **aprobados_algebra_ultima_fecha**. La salida debe verse como:

```
> CANTIDAD DE GENTE QUE SE PRESENTÓ AL EXAMEN: ??
> CANTIDAD DE GENTE QUE APROBÓ EL EXAMEN: ??
> PORCENTAJE DE APROBADOS:  ??.?? %
```

Funciones anónimas

Son funciones que sirven para declarar funciones localmente, como si fuesen una variable. Pueden ser sobreesritas con facilidad.

La sintáxis es la siguiente:

`nombre_funcion = @(lista de argumentos) expresión`

Ejemplo

```
>> potencia = @(vec) sqrt(sum(vec.^2))
```

Eso define una funcion llamada **`potencia()`** que recibe un vector y devuelve la potencia (o energia) de la señal (matemáticamente, la norma euclídea).

```
>> pot = potencia([1 2 3 2.2 1 2 0 1])
pot =  4.9840
```

Funciones anónimas

A diferencia de las funciones normales, las funciones anónimas pueden ver todas las variables declaradas fuera de ellas a la hora de crearse, pero dichos valores quedan fijos una vez creadas. Es decir:

Ejemplo

```
>> k=1.38064852E-23; q=1.60217662E-19;
>> T=25+273;
>> calcIC = @(vbe, Io) Io * exp( vbe / (k*T/q) );
>> I1 = calcIC(0.6, 13e-12)
I1 =  0.18245 %A T=25ºC
>> T=60+273; %Modifico temperatura
>> I2 = calcIC(0.6, 13e-12)
I2 =  0.18245 %No cambia el resultado
```

Function handlers (a.k.a) Punteros a función

Con el símbolo @ antes del nombre de una función, se obtiene una referencia a ella, y puede ser almacenada en una variable.

```
>> seno = @sin;
```

Más aún, esa nueva variable se comporta como una función nueva.

```
>> seno(pi/4)
```

Function handlers (a.k.a) Punteros a función

Es algo común que las funciones reciban referencias a otras funciones como parámetros. Es el caso de `plotyy`.

`plotyy` permite graficar utilizando distintas funciones de ploteo para cada curva. Ejemplo:

Curva con ejes lineal y logarítmico

```
x = 0:0.1:2*pi;
y1 = sin (x);
y2 = exp (x - 1);
plotyy (x, y1, x - 1, y2, @plot, @semilogy);
```