

# Ecuaciones y sistemas lineales

Imanol

11/3/2021

## R

### Sistemas compatibles determinados.

Resolver Sistemas Compatibles Determinados con *solve()*

```
A = rbind(c(1,1,2),c(2,4,-3),c(3,6,-5))
b = c(9,1,0)
AB = cbind(A,b) # Unimos matriz A y b

# Comprobar si es compatible determinado
qr(A)$rank==qr(AB)$rank # Compatible
```

```
## [1] TRUE
```

```
qr(A)$rank==3 # Determinado
```

```
## [1] TRUE
```

```
# Resolver el sistema
solve(A,b)
```

```
## [1] 1 2 3
```

```
# Comprobacion
solution = c(1,2,3)
A%*%solution == b # Confirmado, nos da el vector de terminos independientes.
```

```
##      [,1]
## [1,] TRUE
## [2,] TRUE
## [3,] TRUE
```

Resolver Sistemas Compatibles Determinados con *matlib*

```
A = rbind(c(2,2),c(-1,1))
b = c(1,2)
AB = cbind(A,b) # Unimos matriz A y b

library(matlib)

# Una vez en forma de matriz se puede mostrar
showEqn(A, b)
```

```
## 2*x1 + 2*x2 = 1
## -1*x1 + 1*x2 = 2
```

```
# Calcular rango de la matriz con matlib
R(A)
```

```
## [1] 2
```

```
R(AB)
```

```
## [1] 2
```

```
# Comprobar si el sistema es compatible con matlib
all.equal(R(A),R(AB))
```

```
## [1] TRUE
```

```
# Resolver el sistema con matlib
Solve(A,b, fractions = TRUE) # En este caso es mayúscula.
```

```
## x1 = -3/4
## x2 = 5/4
```

```
# Si fractions es TRUE nos lo muestra en fracciones
```

## Representación de Sistemas.

Se hace uso de la librería `matlib()` y mediante `plotEqn()` si tenemos 2 incógnitas o `plotEqn3()` si tenemos 3 incógnitas.

```
library(matlib)

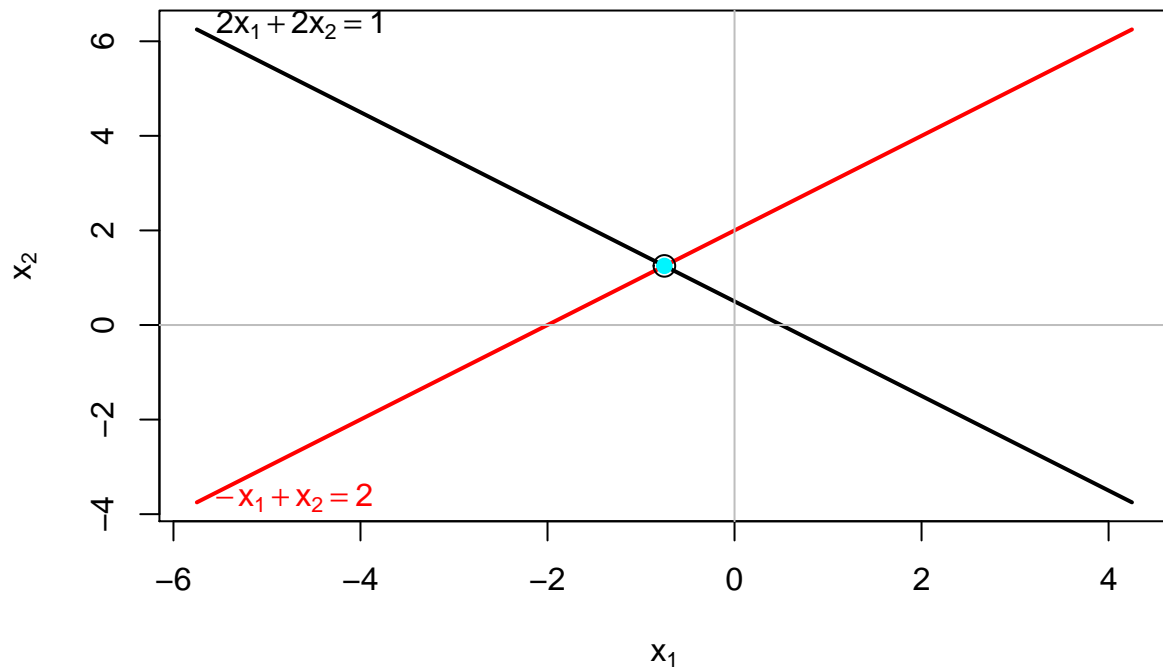
# Representación con plotEqn()

## Dos Incógnitas

### Dos ecuaciones
A = rbind(c(2,2),c(-1,1))
b = c(1,2)
AB = cbind(A,b) # Unimos matriz A y b
plotEqn(A,b)
```

```
## 2*x[1] + 2*x[2] = 1
## -x[1] + x[2] = 2
```

```
points(-3/4,5/4, col = "turquoise1", pch = 19) # Para dibujar puntos con ciertas características
```

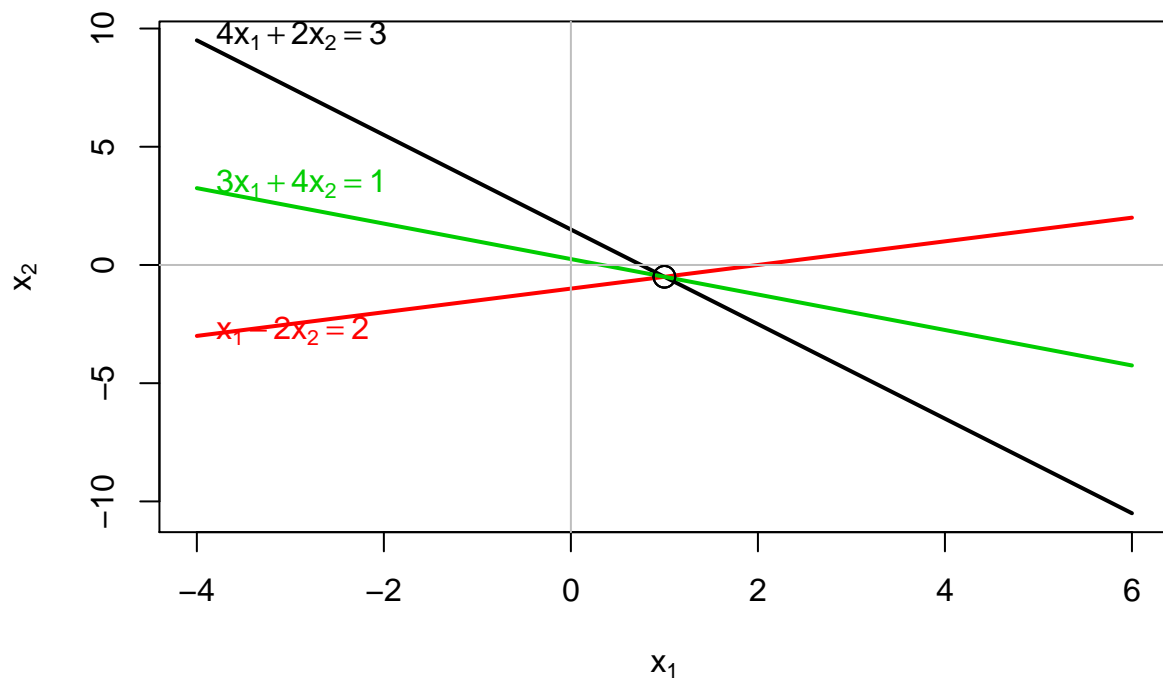


```
#### Tres ecuaciones
A = rbind(c(4,2),c(1,-2),c(3,4))
b = c(3,2,1)
showEqn(A,b)
```

```
## 4*x1 + 2*x2 = 3
## 1*x1 - 2*x2 = 2
## 3*x1 + 4*x2 = 1
```

```
plotEqn(A,b)
```

```
## 4*x[1] + 2*x[2] = 3
## x[1] - 2*x[2] = 2
## 3*x[1] + 4*x[2] = 1
```



*## Tres Incognitas*

```
A = rbind(c(1,1,2),c(2,4,-3),c(3,6,-5))
b = c(9,1,0)
showEqn(A,b)
```

```
## 1*x1 + 1*x2 + 2*x3 = 9
## 2*x1 + 4*x2 - 3*x3 = 1
## 3*x1 + 6*x2 - 5*x3 = 0
```

```
plotEqn3d(A,b, xlim = c(-3,3), ylim = c(0,6))
```

**Tema 2 Ejercicio 1.**

```
A = rbind(c(4,2),c(1,-2),c(3,4))
b = c(3,2,1)
showEqn(A,b)
```

**Solución (a)**

```
## 4*x1 + 2*x2 = 3
## 1*x1 - 2*x2 = 2
## 3*x1 + 4*x2 = 1
```

```
AB = cbind(A,b)

all.equal(R(A),R(AB))
```

```
## [1] TRUE
```

```
# Resolvemos
Solve(A,b,fractions = T)
```

Solución (b)

```
## x1      =      1
##  x2     =  -1/2
##   0     =      0
```

```
#Comprobamos
s = c(1,-1/2)
A%*%s == b # Efectivamente es la correcta
```

```
##      [,1]
## [1,] TRUE
## [2,] TRUE
## [3,] TRUE
```

## Python

Sistemas compatibles determinados.

Resolver Sistemas Compatibles Determinados de manera normal.

```
import numpy as np

A = np.array([[1,1,2],[2,4,-3],[3,6,-5]])
b = np.array([9,1,0])
AB = np.array([[1,1,2,9],[2,4,-3,1],[3,6,-5,0]])

#Comprobar que los rangos son iguales
np.linalg.matrix_rank(A) == np.linalg.matrix_rank(AB)

# Comprobar si ademas el rango es igual a las incognitas
```

```
## True
```

```
np.linalg.matrix_rank(A) == 3

# Resolvemos
```

```
## True
```

```
x = np.linalg.solve(A,b)
x
```

```
# Comprobamos
```

```
## array([1., 2., 3.])
```

```
A.dot(x)
```

```
## array([9., 1., 0.])
```

### Sistemas compatibles determinados con *sympy*

```
from sympy import *
from sympy.solvers.solveset import linsolve

x,y,z = symbols('x,y,z') # Asi se declaran variables en Python
x1,x2,x3 = symbols('x1,x2,x3')
```

```
# Resolver introduciendo la lista de ecuaciones (la facil)
linsolve([2*x1 + 2*x2 - 1, -x1 + x2 - 2], (x1, x2))
```

```
# Resolver introduciendo la matriz aplicada
```

```
## FiniteSet((-3/4, 5/4))
```

```
linsolve(Matrix([[2, 2, 1], [-1, 1, 2]]), (x1, x2))
```

```
#Resolver introduciendo el sistema
```

```
## FiniteSet((-3/4, 5/4))
```

```
AB = Matrix(((2, 2, 1), (-1, 1, 2))) #Matriz ampliada
A = AB[:, :-1] #Todas las filas de todas las columnas menos la ultima
b = AB[:, -1] #Todas las filas de la ultima columna
system = A,b
linsolve(system, x1, x2)
```

```
## FiniteSet((-3/4, 5/4))
```

### Representación de Sistemas.

Se hace uso de la libreria `matplotlib.pyplot`

```
##### Mejor hacer en jupyter que aqui no deja ver graficas
```

```
# Dos ecuaciones con 2 incognitas
```

```
import matplotlib.pyplot as plt
```

```
x1 = np.linspace(0, 10, 100)
```

```
plt.plot(x1, 1/2-x1, x1, (2+x1)/4) # Despejar y
```

```
# Tres ecuaciones con dos incognitas
```

```
## [<matplotlib.lines.Line2D object at 0x000000004527EC10>, <matplotlib.lines.Line2D object at 0x000000004527EC10>]
```

```
x = np.linspace(0, 10, 100)
```

```
plt.plot(x, 3/2-2*x, x, x/2-1, x, (1-3*x)/4) # Despejar y
```

```
# Tres ecuaciones con 3 incognitas
```

```
## [<matplotlib.lines.Line2D object at 0x0000000045291490>, <matplotlib.lines.Line2D object at 0x0000000045291490>, <matplotlib.lines.Line2D object at 0x0000000045291490>]
```

```
from mpl_toolkits.mplot3d import Axes3D
```

```
point1 = np.array([0,0,9/2]) # Un punto valido que cumple la funcion
```

```
n1 = np.array([1,1,2]) # Los valores de la ecuacion
```

```
point2 = np.array([0,0,-1/3]) # Un punto valido que cumple la funcion
```

```
n2 = np.array([2,4,-3]) # Los valores de la ecuacion
```

```
point3 = np.array([0,0,0]) # Un punto valido que cumple la funcion
```

```
n3 = np.array([3,6,-5]) # Los valores de la ecuacion
```

```
D1 = -9 # Termino independiente
```

```
D2 = -1
```

```
D3 = 0
```

```
X, Y = np.meshgrid(range(30), range(30)) # Genero numeros de 0 a 29.
```

```
## Calculamos z en funcion de x e y
```

```
z1 = (-n1[0]*X - n1[1]*Y - D1)*1./n1[2]
```

```
z2 = (-n2[0]*X - n2[1]*Y - D2)*1./n2[2]
```

```
z3 = (-n3[0]*X - n3[1]*Y - D3)*1./n3[2]
```

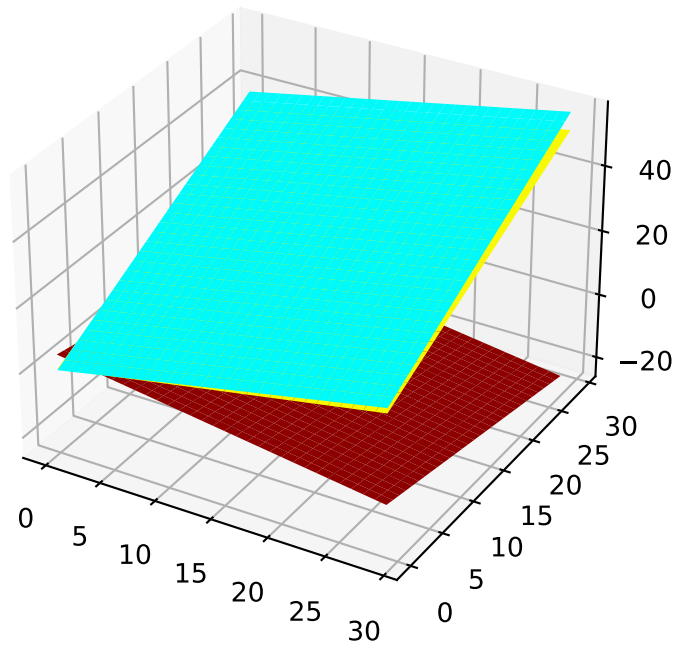
```
plot3d = plt.figure().gca(projection='3d')
```

```
plot3d.plot_surface(X,Y,z1, color='red')
```

```
plot3d.plot_surface(X,Y,z2, color='cyan')
```

```
plot3d.plot_surface(X,Y,z3, color='yellow')
```

```
plt.show()
```



**Metodo de GAUSS.**

No tiene nada

**Sistemas compatibles indeterminados.**

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

A = np.array([[1,1,-1],[1,-1,1],[3,1,-1]])
B = np.array([2,1,5])
AB = np.array([[1,1,-1,2],[1,-1,1,1],[3,1,-1,5]])

# Comprobamos rangos de las matrices
np.linalg.matrix_rank(A)
```

```
## 2
```

```
np.linalg.matrix_rank(A)==np.linalg.matrix_rank(AB)
```

```
# Tambien se puede ver dibujando que tiene infinitas soluciones
```

```
## True
```

```

import matplotlib.pyplot as plt

point1 = np.array([0,0,-2])
n1 = np.array([1,1,-1])

point2 = np.array([0,0,1])
n2 = np.array([1,-1,1])

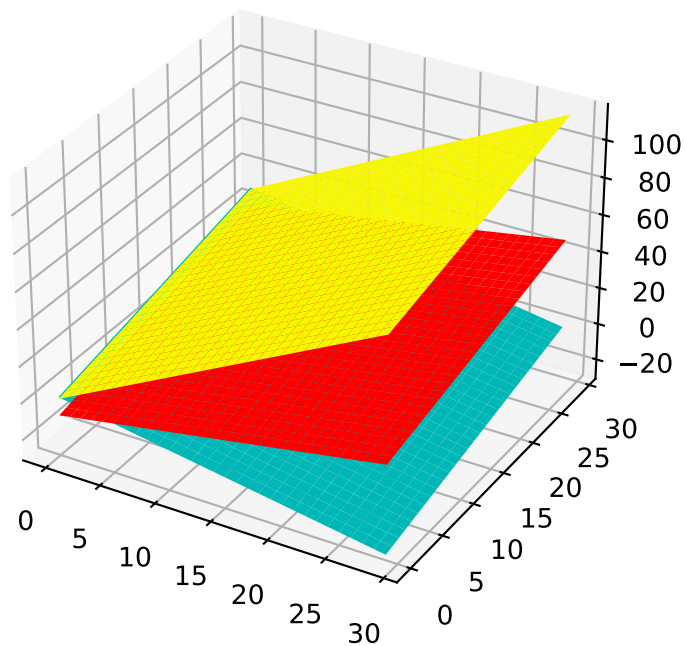
point3 = np.array([0,0,-5])
n3 = np.array([3,1,-1])

X, Y = np.meshgrid(range(30), range(30)) # Genero numeros de 0 a 29.

## Calculamos z en funcion de x e y
z1 = (-n1[0]*X - n1[1]*Y - D1)*1./n1[2]
z2 = (-n2[0]*X - n2[1]*Y - D2)*1./n2[2]
z3 = (-n3[0]*X - n3[1]*Y - D3)*1./n3[2]

## Creamos y dibujamos el plot
plot3d = plt.figure().gca(projection='3d')
plot3d.plot_surface(X,Y,z1, color='red')
plot3d.plot_surface(X,Y,z2, color='cyan')
plot3d.plot_surface(X,Y,z3, color='yellow')
plt.show()

```



## Sistemas Incompatibles.

```
linsolve(Matrix([1, 1, 2], [1, -1, 1], [2, 1, 3])), (x, y))
```

```
## EmptySet
```

Es decir, que el conjunto de soluciones es el conjunto vacío,  $\emptyset$ . Esto se debe a que el sistema es incompatible y no tiene solución.

## Matlab

### Sistemas compatibles determinados.

```
linsolve()
```