Fundamentos Bitcoin

Parte 1: Cryptografía de Curva Elíptica

Bitcoin es una moneda digital que basa su funcionamiento en distintas herramientas criptográficas. Para asegurar la propiedad de un usuario y que éste es el único titular y poseedor de alguna cantidad de Bitcoin se recurre a la criptográfia asimétrica o de llave pública.

Se puede encontrar en Internet noticias que hablan sobre el hackeo y posterior robo de wallets (tanto cold o hot wallets) en bitcoin y lo inseguro que resultan manejar estos. Todas estas notas que pueden o no ser ciertas, no se realizan ni sobre la red ni sobre el protocolo Bitcoin. Todos estos hackeos se deben a brechas en ciberseguridad (un telefóno o computadora con malware) , ingeniería social (pishing) o estafas (hardware wallets maliciosas o intermediarios no confiables) que obtienen la clave privada de la víctima para luego mover esos fondos.

En ningun caso reportado se logró romper el algoritmo de la firma digital. Pues para lograrlo no existen herramientas matemáticas más que la fuerza bruta y es computacionalmente inviable. Por esta razón es muy importante la selección de la clave privada y evitar que sea replicable facilmente (el equivalente a usar 1234 como pin de un celular).

Este tipo de algoritmos y herramientas criptográficas son ampliamente conocidas y estudiadas. Se aplican en distintas tecnologías de uso cotidiano y dependiendo de las características necesarias (como un alto grado de encriptado para un uso militar, por ejemplo) se recurren a distintas herramientas matemáticas. Un estándar usado en distintas industrias desde finanzas hasta almacenamiento es el uso de RSA (que aplica la factorización de números enteros). Un servicio conocido en cloud es MEGA. Esta, por ejemplo, aplica RSA para su cifrado.

alt text

Bitcoin particularmente usa una alternativa a RSA: El algoritmo de firma digital mediante curva eliptica. Este método muestra ser más eficiente en los tamaños de clave e igualmente seguro que RSA. Esta cualidad (el tamaño de la clave) es crítica para la firma masiva de transacciones, pues mientras más espacio en memoria ocupe el blockchain nacesitará mayor hardware de almacenamiento. En partícular Bitcoin utiliza un protocolo que esta definido y estandarizado por el SEC (standards for efficient cryptography): **Secp256k1**.

Esta define la clave con un tamaño único de 160 bits.

Por poner otro ejemplo, los autos eléctricos de Tesla basan su seguridad en un algoritmo de curva elíptica (distinto a Bitcoin) Curve25519 con una clave de 256 bits.

> **NOTA:* La base matemática que usamos:
>
> - Expresiones algebraicas en geometría de curvas planas.
> - Derivadas para el uso de máximo o mínimos para realizar los análisis.
> - Cuerpos fínitos para la representación del sistema.
>
> ---

Curva Elíptica

La siguiente ecuación representa una generalización de la curva elíptica:

$$y^2 = x^3 + Ax + B$$

Esta forma general se denomina Ecuación de Weiestrass.

Proposición

Se busca definir el uso de la Curva Elíptica como un grupo Abeliano, simplificando:

Las operaciones de **SUMA y MULTIPLICACION** deben cumplir la propiedad conmutativa.

$$A + B = B + A$$

$$A * B = B * A$$

Para lo cúal es una condición la existencia de un elemento neutro

$$a + 0 = a$$

Consideramos los puntos P y Q en la curva eliptica (satisfacen su ecuación):

$$P = (x_1, y_1)$$

$$Q = (x_2, y_2)$$

Si trazamos una recta entre ambos puntos, el punto F es la intersección de esta recta con la curva elíptica:

$$F = (x_3, y_3)$$

usamos las expresiones:

$$P + Q = F$$

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = (x_1 - x_3) * \lambda - y_1$$

Si P y Q son distintos puntos $x_1 \neq x_2$

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

y si P y Q son el mismo punto $x_1 = x_2$ y $y_1 \neq 0$

$$\lambda = \frac{3 * x_1^2 + a}{2y_1}$$

La simetría de puntos se da en el eje x:

$$P = (x_1, y_1)$$

$$-P = (x_1, -y_1)$$

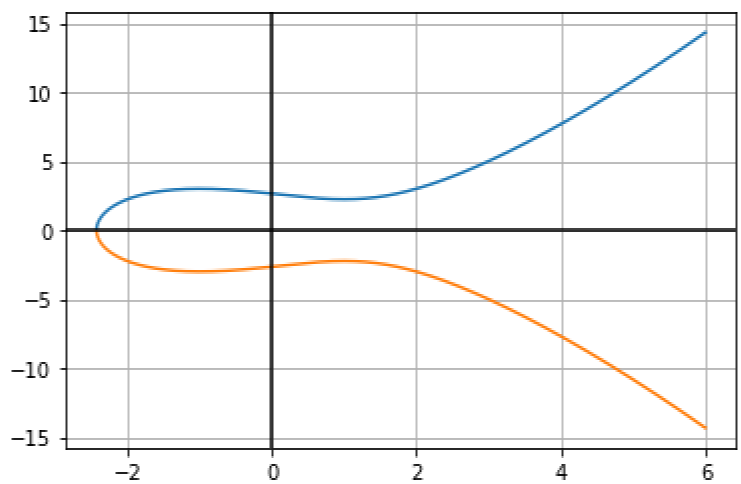
SUMA

Caso 1.

Consideremos la siguiente curva, como un caso partícular de la ecuación de Weiestrass:

$$y^2 = x^3 - 3x + 7$$

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-3*x+7)
plt.plot(x,y,x,-1*y)
plt.grid()
plt.axhline(0,color='black')
plt.axvline(0,color='black');
```



El primer paso de análisis es encontrar el dominio de la variable independiente x. Esto es para asegurarnos que los puntos P y Q que se elijan pertenezcan a la curva y se obtengan números reales al evaluar la función.

> **Bitcoin usa el conjunto de números enteros para evaluar los puntos**. Esto se debe a que un computador para representar un valor con decimales recurre a una notación de punto flotante. Redondeando el número a cierta cantidad de decimales (no se puede representar infinitos decimales en un computador) se genera un error y este se propaga al hacer operaciones como la multiplicación o suma.

Si despejamos la variable 'y' tenemos:

$$y = \pm\sqrt{x^3 - 3x + 7}$$

Los valores dentro del radical deben ser mayor/igual a cero.

$$x^3 - 3x + 7 \geq 0$$

In [ ]:
```
import numpy as np

coeff = [1, 0, -3, 7]
print(np.roots(coeff))
```

```
[-2.42598876+0.j         1.21299438+1.18914511j  1.21299438-1.18914511j]
```

Las raíces que obtenemos rápidamente con numpy son del conjunto de los números complejos. la primera raíz no tiene parte imaginaria, por lo que la podemos tomar como puramente real. Las otras dos raíces son conjugadas, haciendo un poco de álgebra podemos llegar a la siguiente expresión.

$$(x + 2.42)(x^2 - 2.43x + 2.89) \geq 0$$

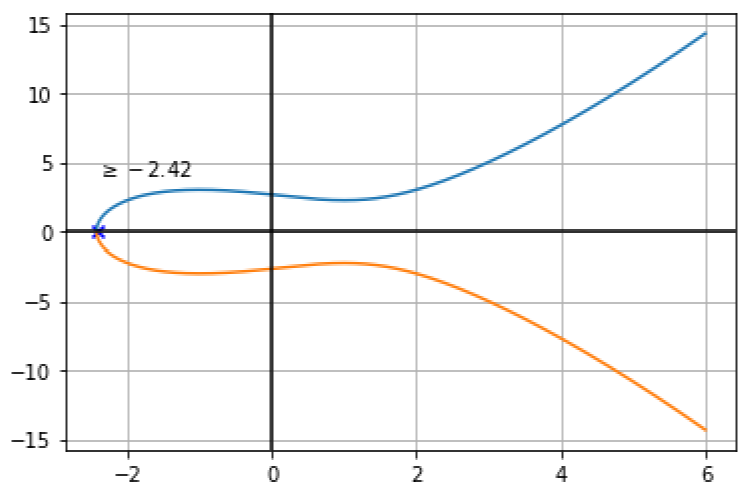
Como se puede apreciar, operar con números redondeados arrastra un error de aproximación.

Siendo el dominio de la variable independiente x:

$$x \geq -2.42$$

In [ ]:

```
x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-3*x+7)
plt.plot(x,y,x,-1*y)
plt.grid()
plt.scatter([-2.42],[0],color='blue',marker='x')
plt.text(-2.42,4,r'$\geq-2.42$')
plt.axhline(0,color='black')
plt.axvline(0,color='black');
```



De esta manera podemos seleccionar puntos P y Q y realizar el cálculo para encontrar su suma $F = P + Q$

Para P:

$$P(x, y) = -2, y$$

Para encontrar el valor correspondiente a 'y' evaluamos la curva en el punto x elegido:

$$y^2 = (-2)^3 - 3 * (-2) + 7 = -8 + 6 + 7 = 5$$

$$y = \sqrt{3}$$

Quedando $P(-2, \sqrt{5})$

Para Q:

$$Q(x, y) = -1, y$$

Para encontrar el valor correspondiente a 'y' evaluamos la curva en el punto x elegido:

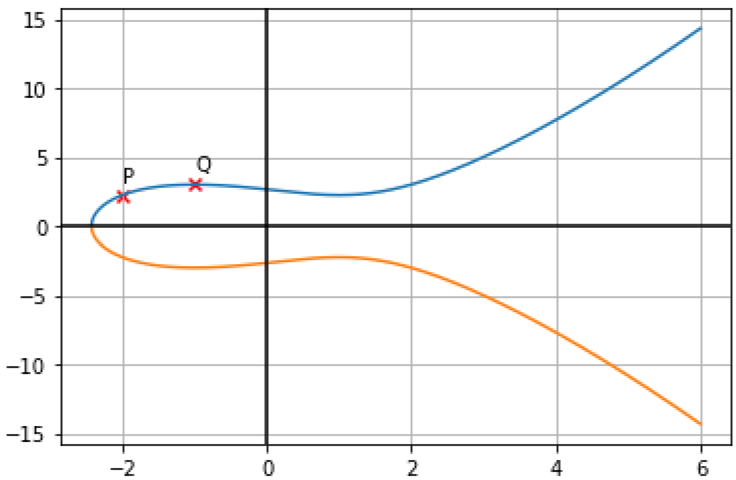
$$y^2 = (-1)^3 - 3*(-1) + 7 = -1 + 3 + 7 = 9$$

$$y = 3$$

Quedando $Q(-1, 3)$

In [ ]:

```
x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-3*x+7)
plt.plot(x,y,x,-1*y)
plt.grid()
plt.scatter([-2,-1],[np.sqrt(5),3],color='red',marker='x')
plt.text(-2,np.sqrt(5)+1,'P')
plt.text(-1,4,'Q')
plt.axhline(0,color='black')
plt.axvline(0,color='black');
```



El punto F es la intersección de la curva con una recta entre los puntos P y Q.

Para encontrar la recta entre dos puntos recordamos geometría analítica:

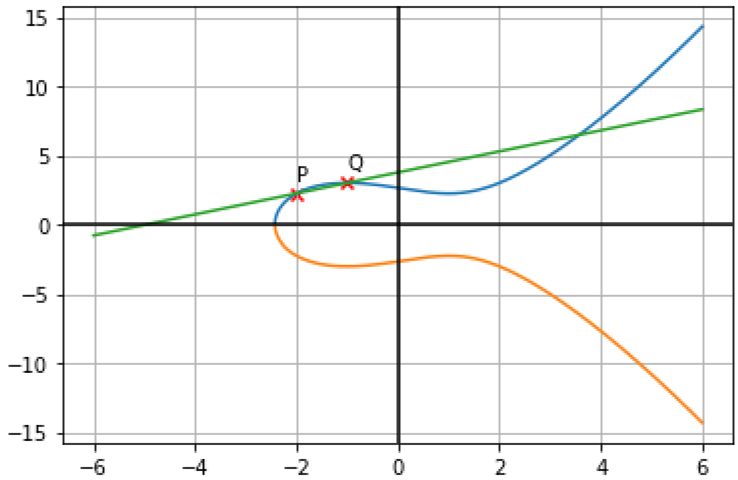
$$y - y_2 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_2)$$

Reemplazando los valores llegamos a la recta:

$$y = (0.76)x + 3.76$$

In [ ]:

```
x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-3*x+7)
#recta
y2=0.76*x+3.76

plt.plot(x,y,x,-1*y)
plt.plot(x,y2)
plt.grid()
plt.scatter([-2,-1],[np.sqrt(5),3],color='red',marker='x')
plt.text(-2,np.sqrt(5)+1,'P')
plt.text(-1,4,'Q')
plt.axhline(0,color='black')
plt.axvline(0,color='black');
```



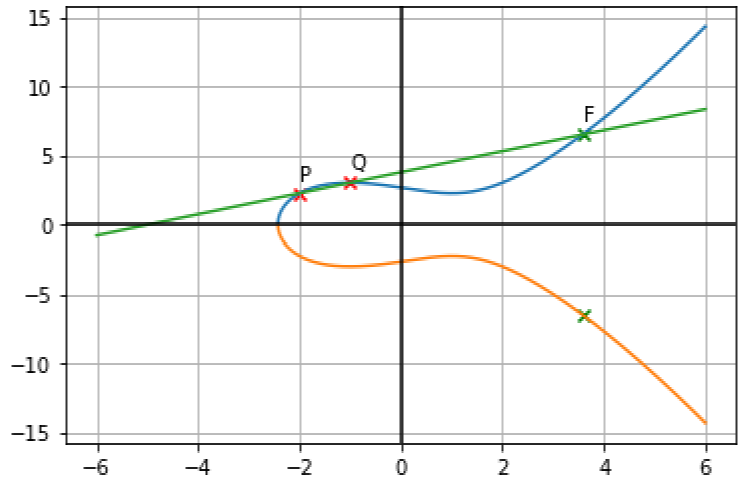
El punto F se puede calcular haciendo el computo de la proposicion de la primera parte.

También se puede obtener si resolvemos las ecuaciones de la recta y la curva, lo que significa una dificultad extra que comparado a la solución de la proposición tiene alto costo computacional.

In [ ]:

```
x1,y1=-2,np.sqrt(5)
x2,y2=-1,3

alpha=(y2-y1)/(x2-x1)

x3=alpha*alpha-x1-x2
y3=(x1-x3)*alpha-y1

x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-3*x+7)
#recta
y2=0.76*x+3.76

plt.plot(x,y,x,-1*y)
plt.plot(x,y2)
plt.grid()
plt.scatter([-2,-1],[np.sqrt(5),3],color='red',marker='x')
plt.text(-2,np.sqrt(5)+1,'P')
plt.text(-1,4,'Q')
plt.axhline(0,color='black')
plt.axvline(0,color='black')

plt.scatter([x3,x3],[y3,-y3],color='green',marker='x')
plt.text(x3,-1*y3+1,'F');
print(x3,-y3)
```

```
3.5835921350012616 6.50155281000757
```



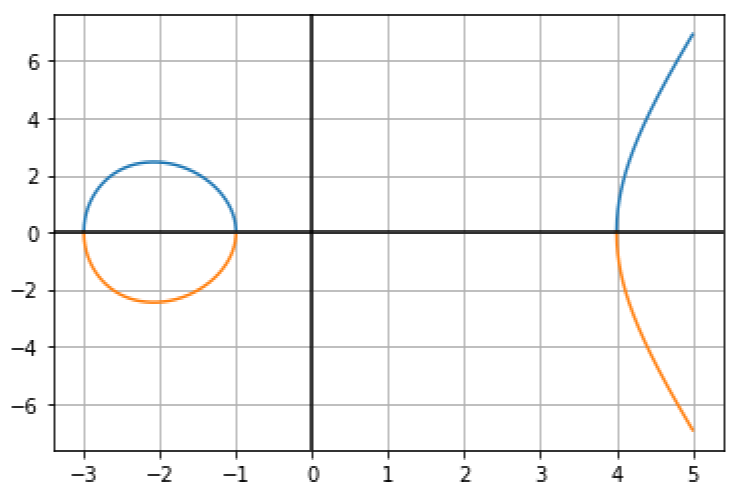
Caso 2.

Otra variante de la ecuación general es:

$$y^2 = x^3 - 13x - 12$$

In [ ]:

```
x = np.arange(-5,5,0.001)
y = np.sqrt(x*x*x-13*x-12)

plt.plot(x,y,x,-1*y)
plt.grid()

plt.axhline(0,color='black')  #horizontal line
plt.axvline(0,color='black');  #horizontal line
```



Para este caso el análisis se realiza de la misma manera.

Si despejamos la variable 'y' tenemos:

$$y = \pm\sqrt{x^3 - 13x - 12}$$

Los valores dentro del radical deben ser mayor/igual a cero.

$$x^3 - 13x - 12 \geq 0$$

In [ ]:

```
import numpy as np
```

```
coeff = [1, 0, -13, -12]
print(np.roots(coeff))
```

```
[ 4. -3. -1.]
```

Factorizando tenemos:

$$(x-4)(x+3)(x+1) \geq 0$$

Con las condiciones: $x \geq 4$,$x \geq -3$ y $x \geq -1$ que se pueden expresar de la siguiente forma:

$D_x = [-3, -1] \cup [4, \infty[$

Los puntos P y Q se escogen de manera que se encuentren en la región izquierda.

$$P(x, y) = -5/2, y$$

Para encontrar el valor correspondiente a 'y' evaluamos la curva en el punto x elegido:

$$y^2 = (-5/2)^3 - 13 * (-5/2) - 12 = 39/8$$

$$y = \sqrt{39/8}$$

Quedando $P(-5/2, \sqrt{39/8})$

$$Q(x, y) = -3/2, y$$

Para encontrar el valor correspondiente a 'y' evaluamos la curva en el punto x elegido:

$$y^2 = (-3/2)^3 - 13 * (-3/2) - 12 = 33/8$$

$$y = \sqrt{33/8}$$

Quedando $Q(-3/2, \sqrt{33/8})$

La recta que pasa por P y Q quedaría:

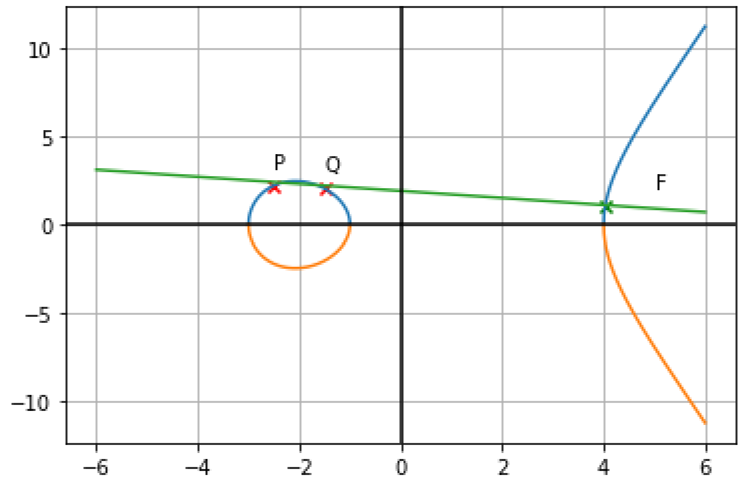
$$y = -0.2 * x + 1.9$$

In [ ]:

```
x1,y1=-5/2,np.sqrt(39/8)
x2,y2=-3/2,np.sqrt(33/8)

alpha=(y2-y1)/(x2-x1)

x3=alpha*alpha-x1-x2
y3=(x1-x3)*alpha-y1

x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-13*x-12)
y2 = -0.2*x + 1.9
plt.plot(x,y,x,-1*y)
plt.plot(x,y2)
plt.grid()
plt.scatter([-5/2,-3/2],[np.sqrt(39/8),np.sqrt(33/8)],color='red',marker='x')
plt.text(-5/2,np.sqrt(39/8)+1,'P')
plt.text(-3/2,np.sqrt(33/8)+1,'Q')
plt.axhline(0,color='black')
```

```
plt.axvline(0,color='black')

plt.scatter([x3],[-y3],color='green',marker='x')
plt.text(x3+1,-y3+1,'F');
print(x3,-y3)
```

```
4.031304442673951 1.0523525020248703
```



Caso 3

Tomamos la misma ecuación anterior y consideramos un punto igual para P y Q. Es decir realizaremos esta operación: $P + P = 2 * P$

Para encontrar la recta tangente al punto P, derivamos la ecuación de la curva elíptica:

$$y^2 = x^3 - 13x - 12$$

despejando y

$$y = \pm\sqrt{x^3 - 13x - 12} = \pm(x^3 - 13x - 12)^{\frac{1}{2}}$$

$$dy = \frac{1}{2}(x^3 - 13x - 12)^{-\frac{3}{2}}(3x^2 - 13)dx$$

Igualamos la derivada de la función de cero para encontrar el punto máximo.

$$y\prime = \frac{1}{2}(x^3 - 13x - 12)^{-\frac{3}{2}}(3x^2 - 13) = 0$$

$$3x^2 - 13 = 0$$

$$x = \pm\sqrt{\frac{13}{3}} = -2.08$$

El resultado que buscamos se encuentra en la región izquierda de la curva, por lo cual el punto x con signo negativo es el valor a usarse. Evaluamos en la función:

$$y^2 = x^3 - 13x - 12$$

$$y^2 = (-2.08)^3 - 13(-2.08) - 12 = -9.02 + 27.06 - 12 = 6.04$$

$$y = \sqrt{6.04} = 2.45$$

Finalmente el punto P:

$$P(x, y) = -2.08, 2.45$$

Para encontrar la recta tangente recurrimos a la siguiente forma:

$$y = mx + b$$

Donde m es la pendiente y representa a la derivada de la curva elíptica evaluada en el punto P.

$$y\prime = m = \frac{1}{2}((-2.08)^3 - 13(-2.02) - 12)^{-\frac{3}{2}}(3(-2.08)^2 - 13)$$

$$m = \frac{1}{2}(24)^{-\frac{3}{2}}(-0.021)$$

$$m = -0.0021$$

Reemplazando en la ecuación de la recta tangente:

$$2.45 = (-0.0021)(-2.08) + b$$

$$b = 2.45$$

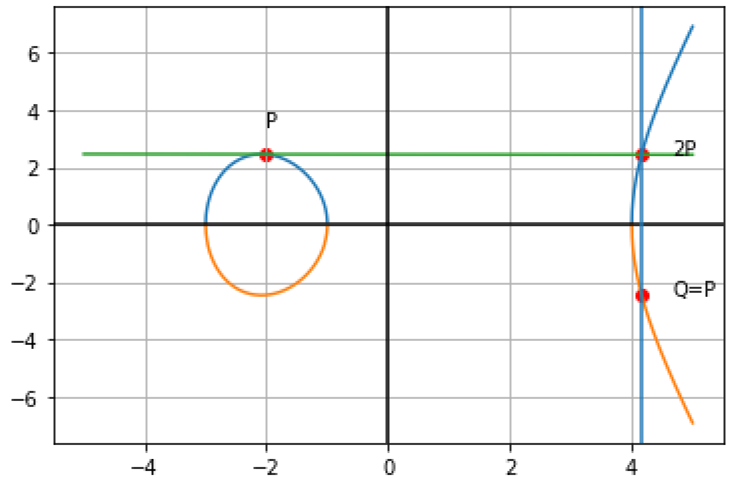
Finalmente:

$$y = (-0.0021)x + 2.45$$

In [ ]:

```
x = np.arange(-5,5,0.001)
y = np.sqrt(x*x*x-13*x-12)
y_2 = (-0.0021)*x+2.45

x1,y1=-2.08,2.45
x2,y2=x1,y1

alpha = (3*x1*x1-13)/(2*y1)

x3=alpha*alpha-x1-x2
y3=(x1-x3)*alpha-y1


plt.plot(x,y,x,-1*y)
plt.plot(x,y_2)
plt.grid()
plt.axhline(0,color='black')  #horizontal line
plt.axvline(0,color='black')  #horizontal line

plt.scatter([-2.02,x3,x3],[2.45,y3,-y3],color='red')
plt.text(-2.02,2.45+1,'P')
plt.text(x3+0.51,y3,'Q=P')
plt.text(x3+0.51,-y3,'2P')
plt.axvline(x3,ymin=-y3,ymax=y3);  #horizontal line
print(x3,y3)
```

```
4.160018019158684 -2.423511760245205
```



Elemento neutro en la suma.

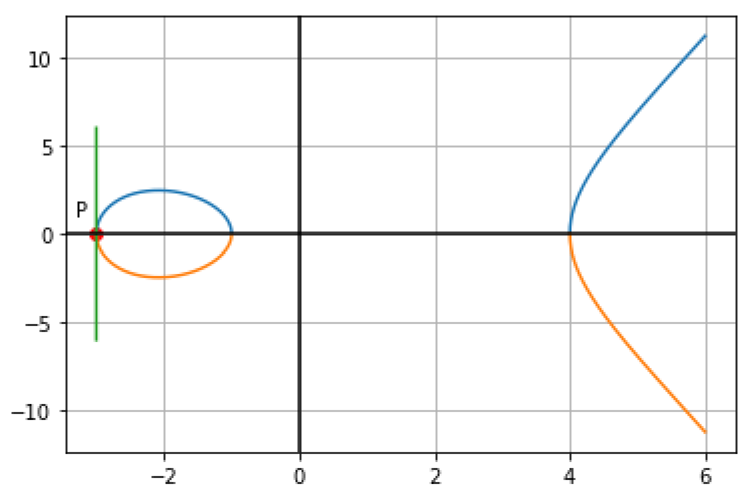
Si la pendiente de la recta no intersecta a la parte derecha del plano coordenado es porque tiene una pendiente de 90 grados (m=0). Es el equivalente a sumar cero pues no cambia la expresión.

In [ ]:

```
x = np.arange(-6,6,0.001)
y = np.sqrt(x*x*x-13*x-12)

y2 = -3*np.ones(len(x))

plt.plot(x,y,x,-1*y)
plt.plot(y2,x)
plt.grid()
plt.axhline(0,color='black')
plt.axvline(0,color='black')
plt.scatter([-3],[0],color='red');
plt.text(-3-0.3,1,'P')
```

Out[ ]:

```
Text(-3.3, 1, 'P')
```



$$P(x1, y1) + 0 = P(x1, y1)$$

Finalmente el elemento neutro junto con la forma de operar la suma en la curva elíptica queda demostrado

Multiplicación

Se denomina un punto P y un número entero natural k

$$k * P = P + \underbrace{\ldots}_{k_{veces}} + P$$

Se define la multiplicación como una suma repetida.

Cuerpos Finitos

Como mencionamos antes un ordenador no puede trabajar con infinita cantidad de números reales. Para lo cual aplicando la matemática modular se asegura que se selecciona un conjunto finito de números a partir de otro infinito.

Para mostrar un ejemplo entendible hablamos de las horas del día. Es un conjunto finito 0,1,2,..,23 por el que se puede representar cualquier número, por decir, 110 hrs. Si partimos de 0 hrs, que hora será transcurridas 110 hrs?

Por intuición se llega a la conclusión: $110hrs = 4 * 24hrs + 14hrs$ Es decir que transcurrieron 4 dias (como una especie de overflow) y 14 hrs.

Una forma de expresarlo es mediante matemática modular: $14 = 110mod24$

De manera que se usa un mod(p) para indicar que la curva se encuentra en un cuerpo finito.

BITCOIN Y LA CURVA ELÍPtica secp256k1

Bitcoin hace el uso de una curva especial que viene definida en el documento normativo Standards for Efficient Cryptography (SEC) (Certicom Research, http://www.secg.org/collateral/sec2_final.pdf) que trabaja con un rango muy grande para su campo finito:

$$P = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

Este número se expresa codificado en hexadecimal p = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F

Si transformamos al sistema decimal corresponde al número: 115792089237316195423570985008687907853269984665640564039457584007908834671663

Secp256k1

Esta se deriva de la ecuación weiestrass con $a = 0$ y $b = 7$:

$$y^2 = x^3 - 7$$

Cuyos valores a y b se expresan en el documento técnico en formato hexadecimal: a = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 b = 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007

El punto baseG en forma comprimida es:

G = 02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 Y en forma no comprimida es: G = 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8

Se puede imaginar como una rejilla con P cuadros por eje. Casi en la misma escala de número de átomos en el universo observable! Pues p es aprox. $p = 1,15.... \times 10^{77}$ y son en promedio 10^{82} átomos.

Por ejemplo se verificara que el siguiente punto (encontrado en la literatura y en la documentación) pertence a esta curva:

$$P(x, y) = (55066263022277343669578718895168534326250603453777594175500187360389116729240, 326$$

In [ ]:

```
p = 115792089237316195423570985008687907853269984665640564039457584007908834671663 #tamaño rejilla
x = 55066263022277343669578718895168534326250603453777594175500187360389116729240
y = 32670510020758816978083085130507043184471273380659243275938904335757337482424
(x ** 3 + 7 - y**2) % p
```

Out[ ]:

```
0
```

Los números P, G son únicos y comunes en el protocolo para que se cumpla la relación modular.

GENERANDO CLAVE PUBLICA

Para generar una clave pública primero es necesario obtener una clave privada de un número k generado aleatoriamente. Luego, esta se múltiplica con una constante llamado punto Generador para producir otro punto en la curva. Este punto es siempre el mismo para todas las claves y esta normado por el éstandar secp256k1.

$$K = k * G$$

K mayúscula es la clave pública k minúscula es la clave privada

La relación K y k es única y se obtiene en un solo sentido. No exite definida una operación DIVISION que asegure encontrar k partiendo de K.

Un ejemplo:

k = 1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD * G

En la siguiente gráfica mostramos como se realiza una suma k veces jugando con la intersección de la recta tangente y su proyección en el eje x

My Image

K = (x, y) donde:

x = F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A

y = 07CF33DA18BD734C600B96A72BBC4749D5141C90EC8AC328AE52DDFE2E505BDB

Para visualizar la multiplicación de un punto y un entero usamos una curva elíptica sobre los números reales Visualizando la multiplicacion de un punto G por un entero k sobre una curva elíptica y muestra el proceso de derivar G, 2G, 4G, como una operación geométrica sobre la curva.

Direcciones Bitcoin

Cuando se desea transferir un valor en bitcoin, solemos encontrar que es la «llave pública» la que se conoce como el 'destinatario' que puede ser libremente compartido (cuando se desea recibir bitcoin). Esta dirección se genera a partir de la clave pública, tiene la siguiente forma (un ejemplo del libro Mastering Bitcoin):

J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy

La dirección es una representación del par clave privada pública o un script de pago. ahora estudiamos el caso particular que la dirección representa y es derivada de una clave pública.

Esta se obtiene por un hash criptográfico de único sentido.

%%%%% aqui se comparte el enlace al post hash %%%%%%

Se aplica un doble hashing a la clave pública: 1 sha256 -> 2 ripemd160

Finalmente se obtiene un hash de 160 bits (20 bytes)

K --> A

Estas direcciones bitcoin se presentan en una códificación especial llamada Base58Check Base58 1 2 3 4 5 6 7 8 9 A B C D E F G H J K L M N P Q R S T U V W X Y Z a b c d e f g h i j k m n o p q r s t u v w x y z + CheckSum

Esto para evitar ambiguedades en la presentación, una representación compacta, menos simbolos necesarios.

public key --> Sha256 --> ripemd160 --> Public Key Hash (20 bytes/160 bits) ---> Base58Check Encode 0x00 prefix (4 bytes extra al final)

El checksum se compara con el de datos y el incluido en el código. Esto evita errores de tipeo evitando mandar transferencias a direcciones incorrectas.

Siendo el producto final Bitcoin Address.

Estos 256 bits se pueden codificar, como vimos, de distintas formas: hexadecimal 4 bytes 64 digitos hexadecimales base58check
wif ()

Wif es un método de seguridad que permite verificar si una dirección es correcta. En el siguiente enlace encontramos más detalle de como se construye

https://en.bitcoin.it/wiki/Wallet_import_format

```
Prefijo base58
```

0x00 dirección bitcoin 1 0x05 dirección Script 3 0x6F dirección testnet m o n 0x80 WIF clave privada 5, K o L
0x0142 BIP38 6P
0x0488B21E BIP32 extendido xpub

Codificando ejemplos

La librería bitcoin para [[python]] originalmente fue escrita por Vitalik Butterin el fundador de Ethereum. Hoy se llama cryptos y es mantenida en comunidad.

se usa

```
pip install cryptos
```

In [ ]:

```python
import cryptos

#generamos una clave privada
valid_private_key = False
while not valid_private_key:
    #El método que se usa para la generación de un número randomico internamente
    #usa RSA, tecnicas mas apropiadas para cryptografia que los paquetes random normales
    private_key = cryptos.random_key()
    print('La llave generada aleatoriamente en HEX: ')
    print(private_key,type(private_key))
    #como dijimos esta parte se puede hacer manualmente al lanzar una moneda 256 veces (bits 1 o 0
    #para su facil lectura se transforma a una codificación hexadecimal

    decoded_private_key = cryptos.decode_privkey(private_key, 'hex')

    print('la llave en formato decimal')
    print(decoded_private_key,type(decoded_private_key))
    #el metodo de 256 lanzamientos puede superar al numero P maximo por lo que se debe
    #verificar que esta en el rango
    print('Rango de números 0 -',cryptos.N)
    valid_private_key =  0 < decoded_private_key < cryptos.N

print('llave final en dec:')
print(decoded_private_key)
```

```
La llave generada aleatoriamente en HEX:
bdbf6a9b9ae60ca2d021d24a76de3e61e2d2824fedeb7518739b33c556bf4d98 <class 'str'>
la llave en formato decimal
85825331951827241751873485306737359194162577756147023214602346186887860800920 <class 'int'>
Rango de números 0 - 115792089237316195423570985008687907852837564279074904382605163141518161494337
llave final en dec:
85825331951827241751873485306737359194162577756147023214602346186887860800920
```

In [ ]:

```python
p_key='1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD'
p_key_num = cryptos.decode_privkey(p_key,'hex')
print('El numero: ',p_key_num)
wif_encoded_private_key = cryptos.encode_privkey(p_key_num, 'wif')
print("Private Key (WIF): ", wif_encoded_private_key)

compressed_private_key = p_key + '01'
print("Private Key Compressed (hex) is: ", compressed_private_key)

wif_compressed_private_key = cryptos.encode_privkey(cryptos.decode_privkey(compressed_private_key,
print("Private Key (WIF-Compressed) is: ", wif_compressed_private_key)



# Multiply the EC generator point G with the private key to get a public key point
public_key = cryptos.fast_multiply(cryptos.G, p_key_num)
print("Public Key (x,y) coordinates is:", public_key)
# Encode as hex, prefix 04
hex_encoded_public_key = cryptos.encode_pubkey(public_key,'hex')
print("Public Key (hex) is:", hex_encoded_public_key)
# Compress public key, adjust prefix depending on whether y is even or odd
(public_key_x, public_key_y) = public_key
if (public_key_y % 2) == 0:
```

```
    compressed_prefix = '02'
else:
    compressed_prefix = '03'

hex_compressed_public_key = compressed_prefix + cryptos.encode(public_key_x, 16)
print("Compressed Public Key (hex) is:", hex_compressed_public_key)
# Generate bitcoin address from public key
print("Bitcoin Address (b58check) is:", cryptos.pubkey_to_address(public_key))
# Generate compressed bitcoin address from compressed public key

print("Compressed Bitcoin Address (b58check) is:", cryptos.pubkey_to_address(hex_compressed_public
```

```
El numero:  13840170145645816737842251482747434280357113762558403558088249138233286766301
Private Key (WIF):  5J3mBbAH58CpQ3Y5RNJpUKPE62SQ5tfcvU2JpbnkeyhfsYB1Jcn
Private Key Compressed (hex) is:  1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
01
Private Key (WIF-Compressed) is:  KxFC1jmwwCoACiCAWZ3eXa96mBM6tb3TYzGmf6YwgdGWZgawvrtJ
Public Key (x,y) coordinates is: (108626704259373488493324494832963472198167093861790438979212875284973843395610, 3532285151429480400098290360892322426461524297929807392099748929454186978267)
Public Key (hex) is: 04f028892bad7ed57d2fb57bf33081d5cfcf6f9ed3d3d7f159c2e2fff579dc341a07cf33da18b
d734c600b96a72bbc4749d5141c90ec8ac328ae52ddfe2e505bdb
Compressed Public Key (hex) is: 03f028892bad7ed57d2fb57bf33081d5cfcf6f9ed3d3d7f159c2e2fff579dc341a
Bitcoin Address (b58check) is: 1424C2F4bC9JidNjjTUZCbUxv6Sa1Mt62x
Compressed Bitcoin Address (b58check) is: 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
```

In [ ]:

```
# Convirtiendolo a formato WIF WALLET IMPORT FORMAT

wif_encoded_private_key = cryptos.encode_privkey(decoded_private_key, 'wif')
print("Private Key (WIF): ", wif_encoded_private_key)
```

```
Private Key (WIF):  5KFrS2iGSHxqWi5eNSPcWDEheZeekZi5UbBkaqC7348ybeai4oG
```

In [ ]:

```
# Add suffix "01" to indicate a compressed private key
compressed_private_key = private_key + '01'
print("Private Key Compressed (hex) is: ", compressed_private_key)
```

```
Private Key Compressed (hex) is:  bdbf6a9b9ae60ca2d021d24a76de3e61e2d2824fedeb7518739b33c556bf4d98
01
```

In [ ]:

```
# Generate a WIF format from the compressed private key (WIF-compressed)

wif_compressed_private_key = cryptos.encode_privkey(cryptos.decode_privkey(compressed_private_key,
print("Private Key (WIF-Compressed) is: ", wif_compressed_private_key)
```

```
Private Key (WIF-Compressed) is:  L3aZBUZ98dsT88KVwd4ohw1UQo7BdHNJkmtTNKi1e4rCoge6JGCa
```

In [ ]:

```
# Multiply the EC generator point G with the private key to get a public key point
public_key = cryptos.fast_multiply(cryptos.G, decoded_private_key)
print("Public Key (x,y) coordinates is:", public_key)
# Encode as hex, prefix 04
hex_encoded_public_key = cryptos.encode_pubkey(public_key,'hex')
print("Public Key (hex) is:", hex_encoded_public_key)
# Compress public key, adjust prefix depending on whether y is even or odd
(public_key_x, public_key_y) = public_key
if (public_key_y % 2) == 0:
    compressed_prefix = '02'
else:
    compressed_prefix = '03'

hex_compressed_public_key = compressed_prefix + cryptos.encode(public_key_x, 16)
print("Compressed Public Key (hex) is:", hex_compressed_public_key)
```

```python
# Generate bitcoin address from public key
print("Bitcoin Address (b58check) is:", cryptos.pubkey_to_address(public_key))
# Generate compressed bitcoin address from compressed public key

print("Compressed Bitcoin Address (b58check) is:", cryptos.pubkey_to_address(hex_compressed_public
```

```
Public Key (x,y) coordinates is: (851798347089961496895587262977681158755782150399146478742831068
35621994286768, 87770814839697955054470426787546516324375888302768512071143864821455954928343)
Public Key (hex) is: 04bc5213f42be513b6791f7e3227761713629a83f706cd779b266c7c4eb31fceb0c20c854e8ae
432f8458bc81aa617b68618de45ae64dea56c1ca80834fb9062d7
Compressed Public Key (hex) is: 03bc5213f42be513b6791f7e3227761713629a83f706cd779b266c7c4eb31fceb0
Bitcoin Address (b58check) is: 1EQh1T3VpGcysZDx4zZ8UyjMudnBVu11sZ
Compressed Bitcoin Address (b58check) is: 1GMn1mLMwrTfW23KnvHM68d6cFPiQ9Bwj
```

In [ ]: