plot2_lecture

October 23, 2023

1 Att rita grafer (plot)

matplotlib.pyplot. Det kallas att plotta med Pyplot. Det ger snabbt resultat. Om man vill kontrollera ritandet mer i detalj så rekommenderar python.org att man använder det objekt orienterade skrivsättet i stället. Vi behandlar inte det här utan det kommer att finnas på en annan plats.

Här på jupyter ritas graferna på ett visst sätt med de inställningar som finns. Ritar du dem på replit.com eller på din egen dator kommer det att se annorlunda ut.

1.1 Mallen

För att kunna rita grafer måste vi importera moduler. Modulen numpy innehåller något som liknar listor men passar grafritning bättre.

med en array kan man göra samma operation på alla elementen i listan. Man kan kvadrerar varje element för sig

Vi börjar med att skapa x-värden

[6]:
```
import numpy as np


Lx=[1,2,3]
#Ly=Lx+[3]
#print(Ly)

#Lx=np.array(Lx)
#Ly=Lx+3
#print(Ly)

Lx=np.array(Lx)
Ly=Lx*Lx
print(Ly)


#Lx = np.array([1,2,4,5])
#Lysin=np.sin(Lx)
#print("Lx= ", Lx, ",  sin för Lx= ", Lysin)

#Lykvadrat=Lxny**2
```

```
#print("array")
#print("Lx= ", Lx, ",  Lx i kvadrat= ", Lykvadrat)
```

```
[1 4 9]
```

Man har gjort det ännu smidigare genom kommandon som man infört. Den heter `np.linspace()` och gör arrayer.

[2]:
```
import numpy as np

L1 = np.linspace(1,4,7)  # array, (start, stopp, antal)
print(L1)
print(2*L1, type(L1))  # array
```

```
[1.  1.5 2.  2.5 3.  3.5 4. ]
[2. 3. 4. 5. 6. 7. 8.] <class 'numpy.ndarray'>
```

[9]:
```
# Även range går förstås
L1=np.array(range(1,11,1))
print(type(L1),L1)
```

```
<class 'numpy.ndarray'> [ 1  2  3  4  5  6  7  8  9 10]
```

ndarray står för n-dimensional array, en datatyp.

Skrivsättet blir väldigt suggestivt. När man ska skriva det som behövs för en grafritning.

[18]:
```
import numpy as np

x=np.linspace(-np.pi,np.pi,6)
print(x)
```

```
[-3.14159265 -1.88495559 -0.62831853  0.62831853  1.88495559  3.14159265]
```

Kommandot `linspace()` ska ha start och stopp och antal värden `linspace(start, stopp, num)`. I detta fall har jag använt intervallet från -2π till 2π. Allra sist står antalet (`num`) tal som jag vill ha. I detta fall 10 stycken.

Observera att stopp ingår, se nedan.

[3]:
```
import numpy as np

x=np.linspace(-2,2,10)
print(x)
```

```
[-2.         -1.55555556 -1.11111111 -0.66666667 -0.22222222  0.22222222
  0.66666667  1.11111111  1.55555556  2.        ]
```

Vi gör en värdetabell, både x och y

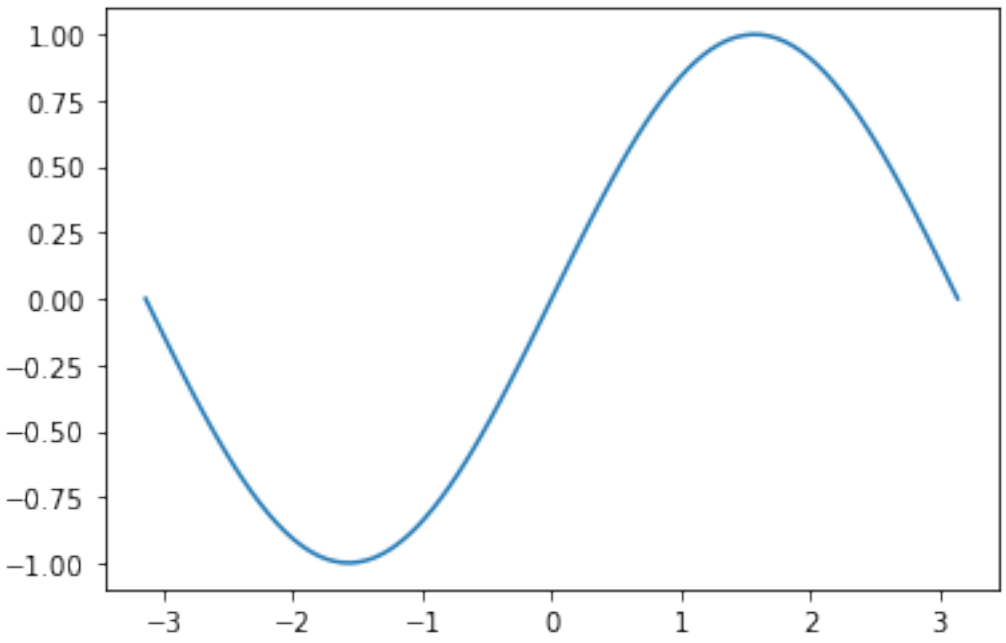
[10]:
```
import numpy as np

x = np.linspace(-np.pi,np.pi,10)
print("x-värden",x)
y = np.sin(x) #nästan samma skrivsätt som i matematiken
print("y-värden",y)
```

```
x-värden [-3.14159265 -2.44346095 -1.74532925 -1.04719755 -0.34906585
0.34906585
  1.04719755  1.74532925  2.44346095  3.14159265]
y-värden [-1.22464680e-16 -6.42787610e-01 -9.84807753e-01 -8.66025404e-01
 -3.42020143e-01  3.42020143e-01  8.66025404e-01  9.84807753e-01
  6.42787610e-01  1.22464680e-16]
```

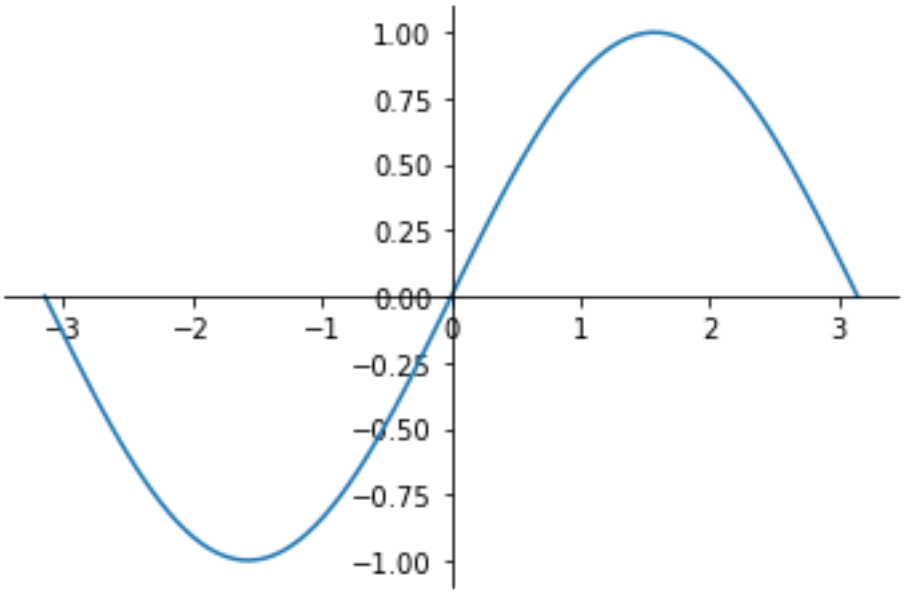
Nu har vi vår värdetabell klar. Nu behöver vi ladda en modul som sköter om själva plottandet.

Modulen som behöver importeras, heter `matplotlib.pyplot`. De nya kommandona framgår av koden.

[8]:
```
# Mall för enkel plottning

import matplotlib.pyplot as plt
import numpy as np

# Värdetabell
x = np.linspace(-np.pi, np.pi, 100)
y = np.sin(x)

#Nu ritar vi!
plt.plot(x,y)
plt.show()
```



[12]:
```
"""
import matplotlib.pyplot as plt
import numpy as np


ax = plt.gca()
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')

# Mall för enkel plottning

# Värdetabell
x = np.linspace(-np.pi, np.pi, 100)
y = np.sin(x)

#Nu ritar vi!
plt.plot(x,y)
plt.show()
"""
```



Python justerar själv koordinatsystemet om man inte säger något, dock finns det kommando för det också.

Om du skrivit raderna i en textfil, tex. på repl.it måste du även ange kommandot `plt.show()` annars får du ingen graf att själv titta på. (jupyter klarar sig utan den, men lägger till en rad i utskriften om den inte är med)

Följden av kommandon (import, x=, y=, plot…) används ofta som en mall, sedan är det bara att fylla i och ändra efter behov.

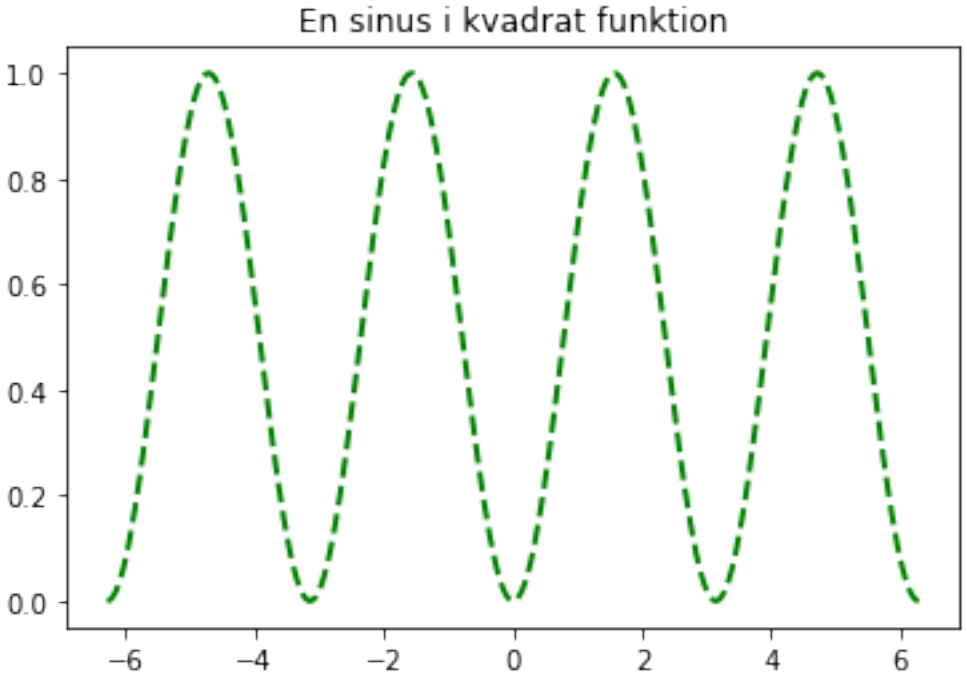
1.2 Utsmyckning

Naturligtvis kan man själv ändra koordinatsystem, rita flera kurvor i samma diagram, ändra färger på kurvorna, lägga in förklarande text osv.

Exempel följer.

[17]:
```
import matplotlib.pyplot as plt
import numpy as np

# Värdetabell
x = np.linspace(-2*np.pi, 2*np.pi, 300)
y = np.sin(x)*np.sin(x)

#Rita med utsmyckning
plt.plot(x,y, color="green",
         linestyle='dashed',
         #marker='o',
         #markerfacecolor='blue',
```

```
        markersize=12,
        linewidth=2)

plt.title('En sinus i kvadrat funktion')

plt.show()
```



Observera hur hela kommandot plot() är uppdelat på flera olika rader, det ökar läsbarheten. Det är tillåtet att göra så och python väntar på den avslutande högerparentesen.

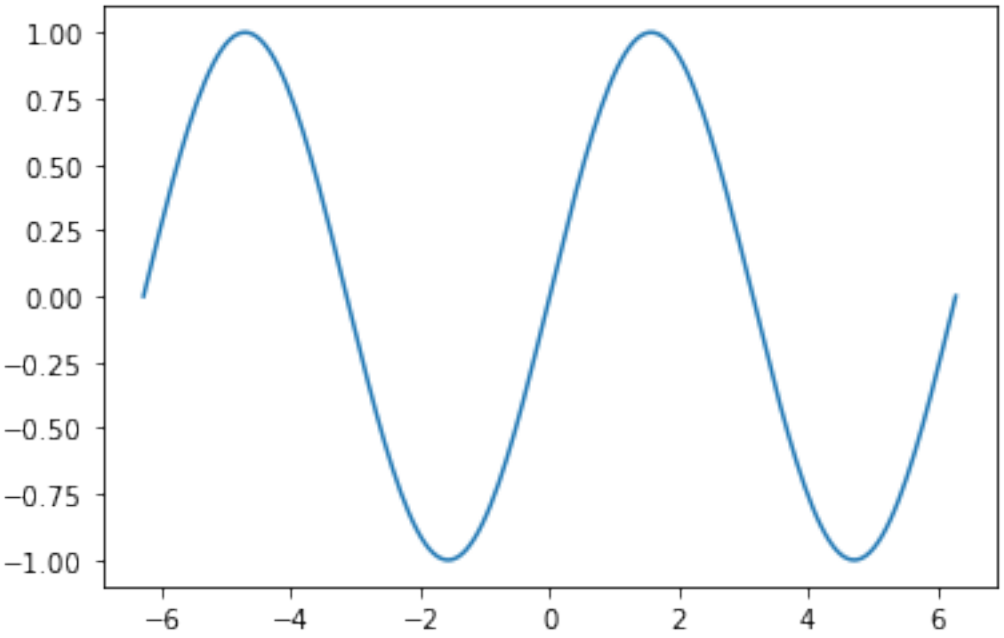
- `color="green"` betyder att kurvan ritas med grön färg
- `linestyle="dashed"` betyder att kurvan är streckad
- `marker="o"` innebär att datapunkterna markeras med en cirkelskiva
- `markerfacecolor="blue"` gör att cirklarna är fyllda med blått (inte gröna som kurvan)
- `markersize=12` anger storleken på markören
- `linewidth=6` anger bredden på kurvan

  Solid Dashed Dotted Dashdot None

Som du kanske inser från tidigare avsnitt så har flera av parametrarna (color, linestyle osv) egna default (förinställda) värden som används om du inte anger något. Vi har tex. anget att color ska ha värdet “green”, men om vi inte skickar argumentet “green” så finns det ett förinställt värde som python använder.

Experimentera. Observera stavningen av färg, det är color inte colour. Observera också att det är strängar som ska anges så citattecken måste vara med. Som På replit.com finns det möjlighet att spara figuren genom att klicka på en ikon i figuren. Men man kan också skriva till ett kommando för det `plt.savefig()`.

[12]:
```
import matplotlib.pyplot as plt
import numpy as np


x=np.linspace(-2*np.pi, 2*np.pi, 200)
y=3*np.sin(4*x)
plt.plot(x,y)

plt.savefig('plot.png') # jpg, pdf
plt.show()
```



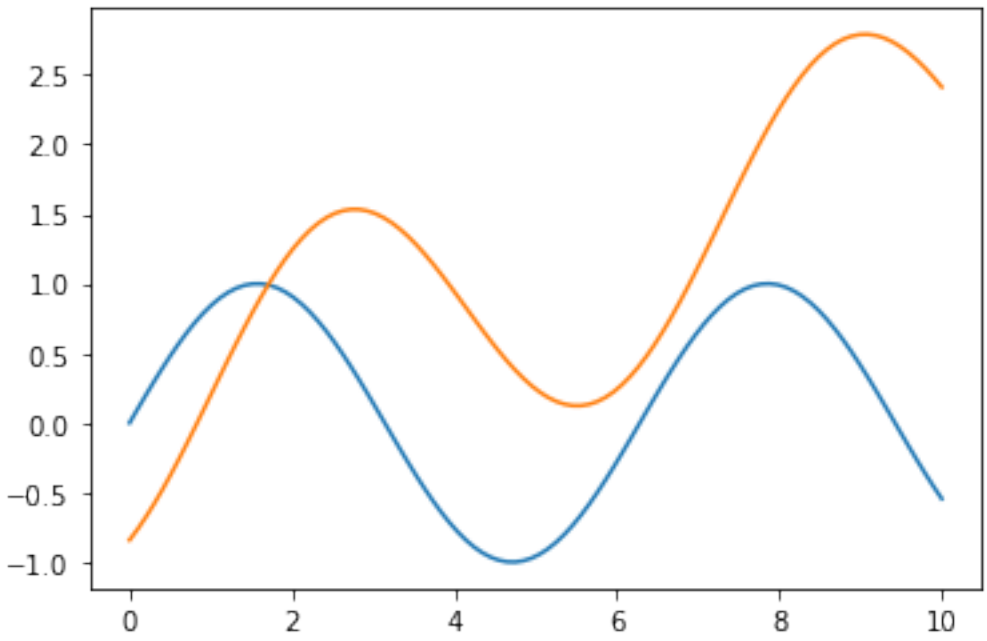
På replit.com syns grafen till höger (i standardinställningen) och en fil med namnet plot.png skapas och läggs bland filerna till vänster där main.py ligger. Denna kan du sedan ladda ner. Flera olika bildformat fungerar.

1.3 Två kurvor i samma graf

Vi har ritat endast en kurva i grafen, nu ritar vi 2.

[4]:
```
import matplotlib.pyplot as plt
import numpy as np
```

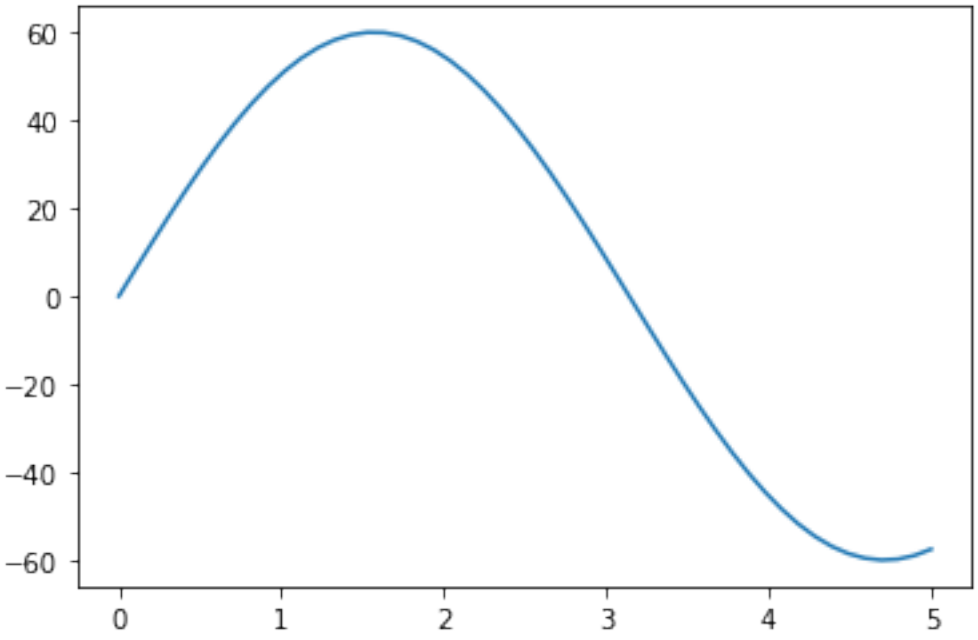
```
# Kurva 1
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x,y)

# Kurva 2
x = np.linspace(0, 10, 100)
y = np.sin(x-1)+ 0.2*x
plt.plot(x,y)

#Ritas i samma graf!

plt.show()
```

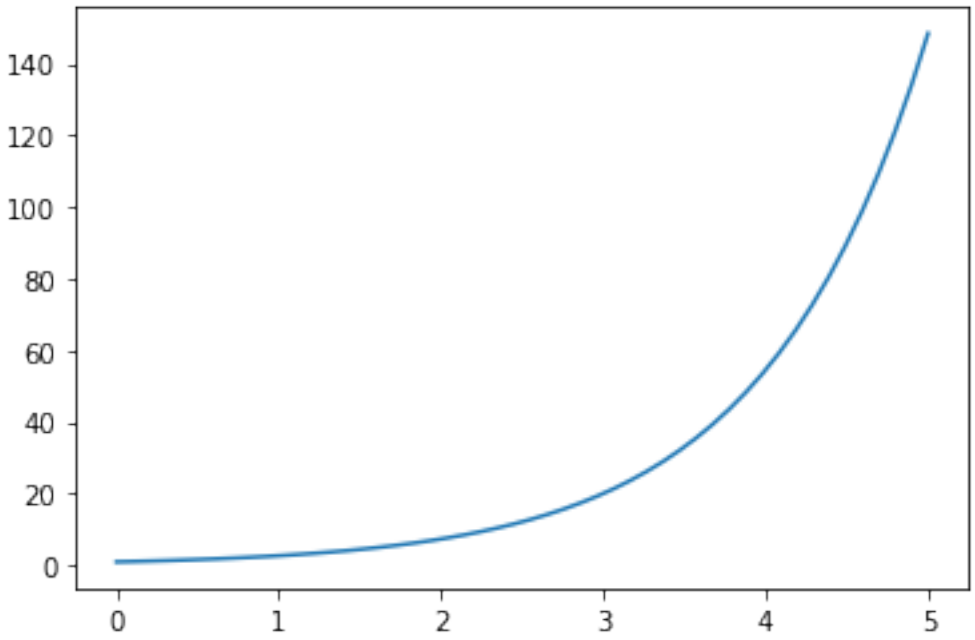


Vad är det som gör att det blir två kurvor i en graf och inte två grafer med en kurva i varje? Det som är signifikant är att kommandot plt.show() kommer efter att vi gett två kommandon plt.plot(). Om vi ger kommandot plt.show() omedelbart efter ett plt.plot() så får vi två grafer.

[5]:
```
import matplotlib.pyplot as plt
import numpy as np

# Kurva 1
x = np.linspace(0, 5, 50)
y = 60*np.sin(x)
plt.plot(x,y)
plt.show()  #!!!!!!!!!!!
```

```
# Kurva 2
x = np.linspace(0, 5, 50)
y = np.exp(x)
plt.plot(x,y)
plt.show()  #!!!!!!!!!!!
```



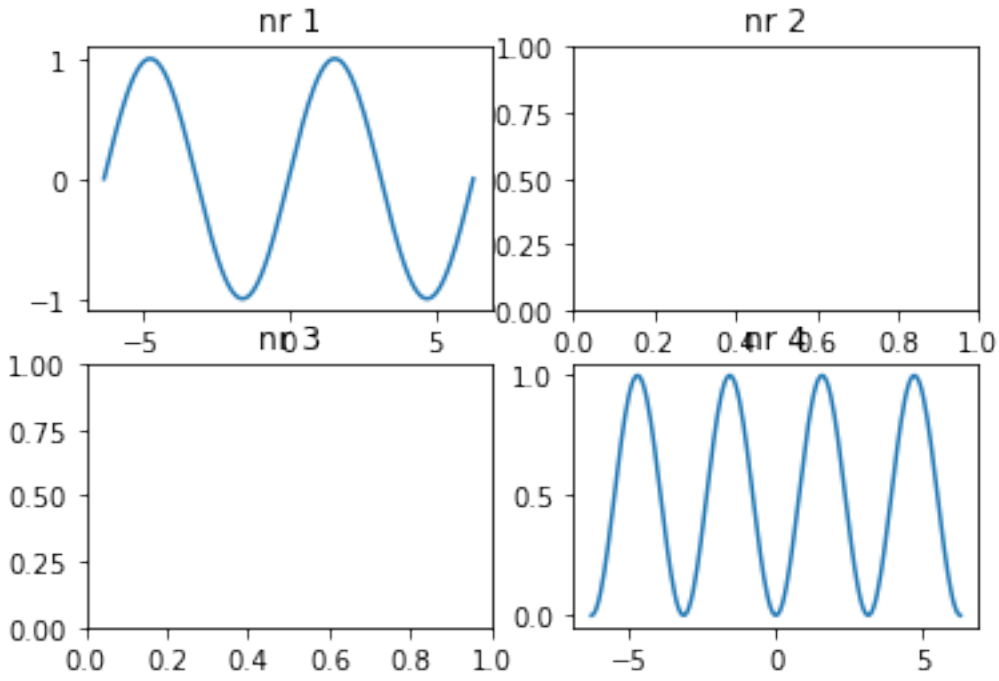


1.4 Staplade grafer

Rita flera grafer sidan om varandra.

För att kunna rita så kallade subplots behövs en styrning av var graferna ska placeras. Man tänker i rader och kolumner. Kommandot är `subplot(nrows, ncols, index)`. `nrows` anger antalet rader och `ncols` anger antalet kolumner. Index anger var grafen ska ritas. Index går från vänster till höger och uppifrån och ner.

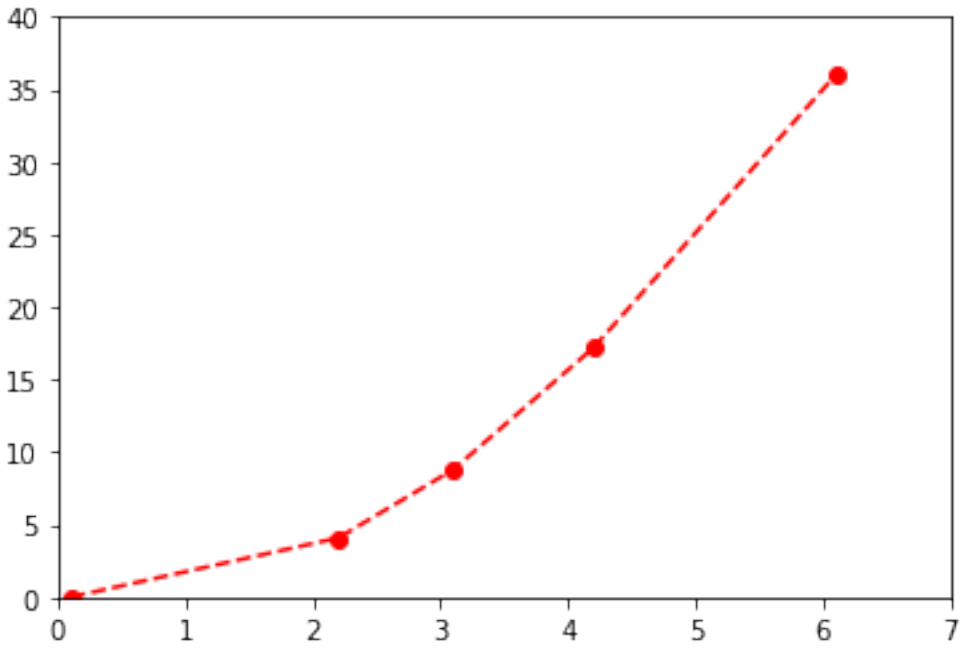
[18]:
```
"""
Rita båda funktionerna i två olika grafer samtidigt

subplot(nrows, ncols, index)

"""

import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-2*np.pi, 2*np.pi, 200)
y = np.sin(x)
plt.subplot(2,2,1) # Två rader, två kolumn, index 1
plt.title("nr 1")
plt.plot(x,y)


x = np.linspace(-2*np.pi, 2*np.pi, 200)
y = np.sin(x)*np.sin(x)
plt.subplot(2,2,4) # index 4
plt.title("nr 4")
plt.plot(x,y)

plt.subplot(2,2,2) # index 2
plt.title("nr 2")

plt.subplot(2,2,3) # index 3
plt.title("nr 3")

#plt.tight_layout()


plt.savefig('plot2.png')
plt.show()
```



1.5 Mätdata

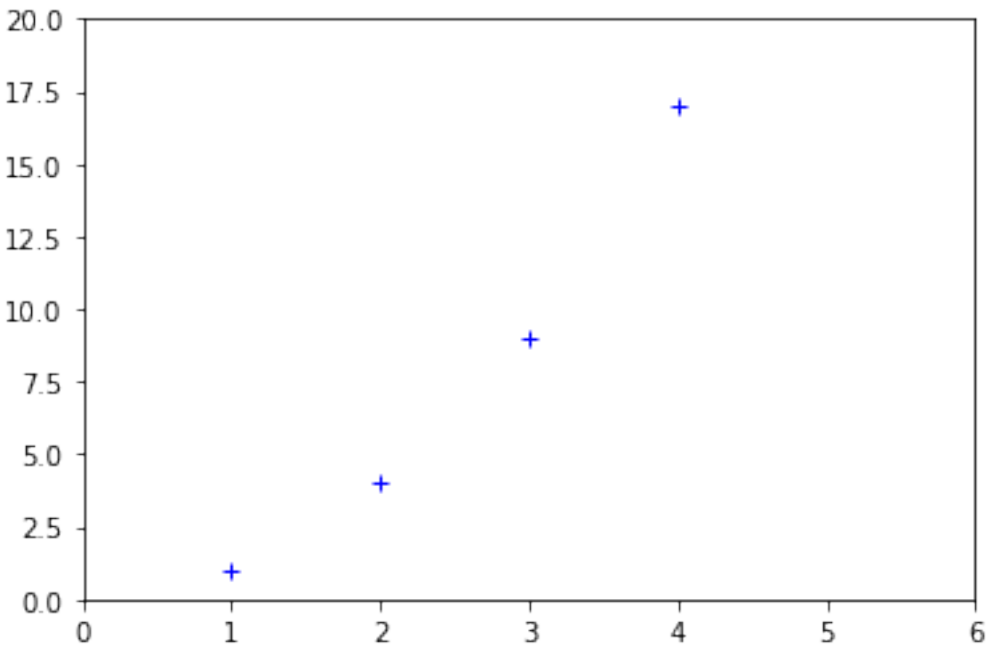
Vi använder mätdata från ett experiment och skriver den i listor. Axlarna bestämmer vi själva denna gång: `plt.axis([0,6,0,20])` anger att x-axeln går från 0 till 6 och y-axeln från 0 till 20. Argumentet är en lista.

[24]:
```
import matplotlib.pyplot as plt
import numpy as np

# Mätdata som skrivits in i listor
x = [0.1, 2.2, 3.1 ,4.2, 6.1]
y = [0.02, 4.1, 8.8, 17.2, 36.0]
plt.axis([0,7,0,40]) # Observera att det är en lista.

plt.plot(x, y, 'ro--')
#röd cirkel, streckad kurva

plt.show()
```



Vill vi inte ha de räta linjerna mellan datapunkterna så heter commandot `plt.scatter(x,y)`. Man kan också utelämna typen av kurva `plt.plot(x, y, 'ro')` så att `--` saknas.

[16]:
```
import matplotlib.pyplot as plt
import numpy as np

# Mätdata som skrivits in i listor
x = [1,2,3,4]
y = [1,4,9,17]
plt.axis([0,6,0,20])

# Nytt kommando! Två alternativ.
# plt.scatter(x, y)

# Eller detta
plt.plot(x, y, 'b+') # blue för blå, markör är + tecken

plt.show()
```



[ ]:

1.6 Teckenförklaring (legend)

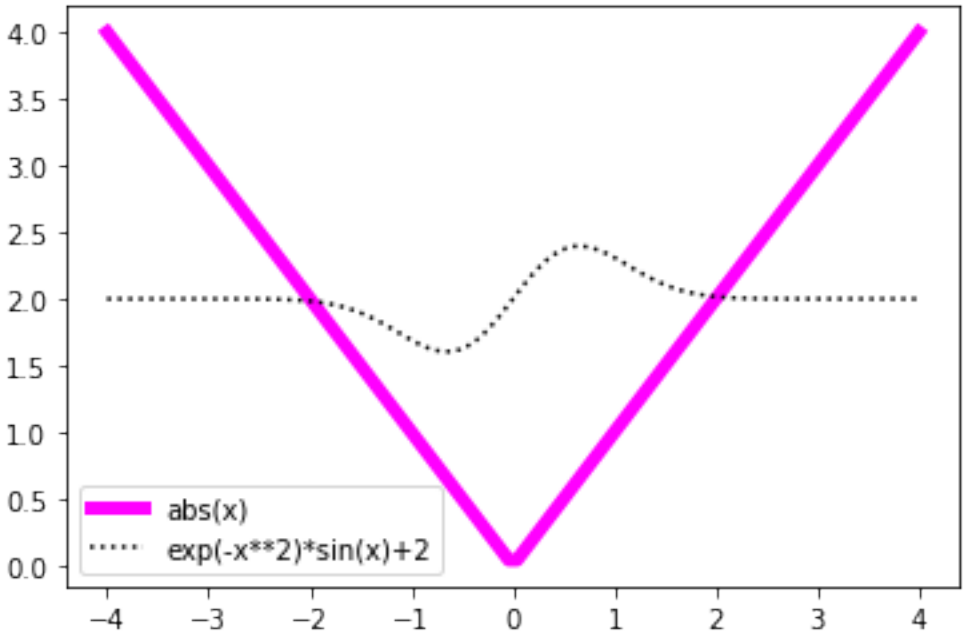
Om flera kurvor ritas kan det vara bra att ge information om dem. I detta fall låter vi python bestämma var det finns plats. Kommandona som används är:

`plot(x,y, label='abs(x)')` , `plot(x,y,'k:', label='exp(-x)*sin(x)')` Dvs. vi anger en `label()` för att kunna särа på kurvorna.

Teckenförklaringen: `legend(loc='best', fontsize='small')`

[27]:
```
import numpy as np
import matplotlib.pyplot as plt

# abs(x)
x = np.linspace(-4, 4, 100)
y = abs(x)
#plt.plot(x,y,"g", label='abs(x)')
plt.plot(x,y, color="magenta", linewidth="5", label='abs(x)')

# exp(-x**2)*sin(x)
x = np.linspace(-4, 4, 100)
y = np.exp(-x**2)*np.sin(x)+2
plt.plot(x,y,'k:', label='exp(-x**2)*sin(x)+2')  # svart (black) och pricklinje␣
 ↪(dotted) linje
```

```
plt.legend(loc='best', fontsize='medium')

plt.show()
#savefig('plot.png')
```

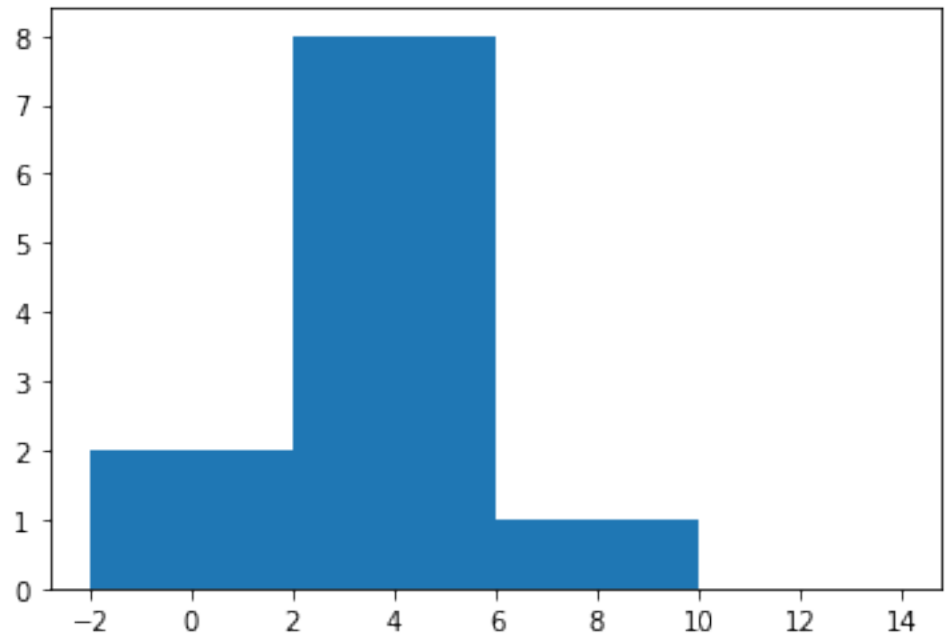


1.7 Histogram

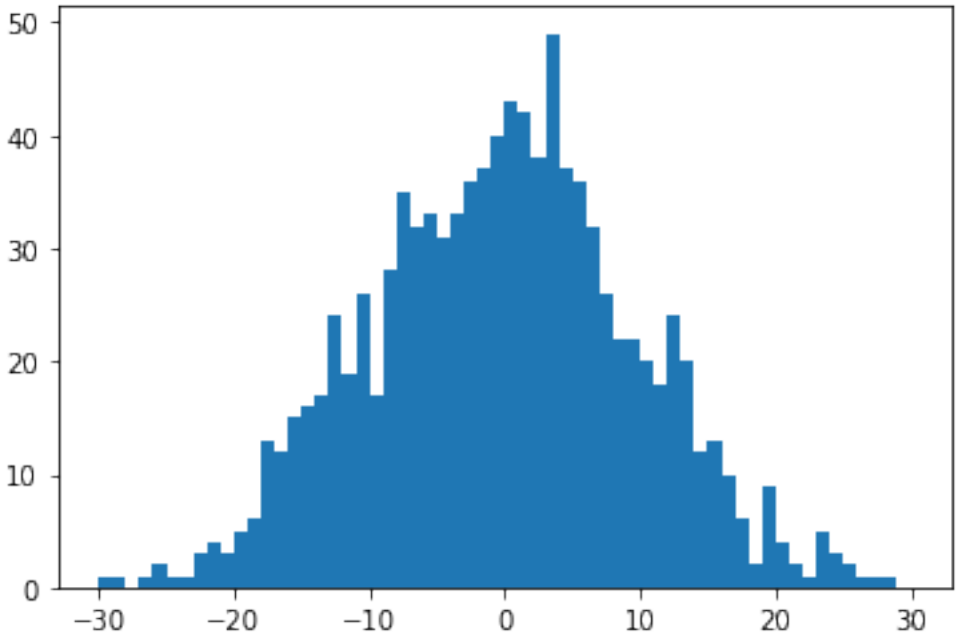
Även histogram kan ritas med matplot.pyplot.

[37]:
```
import matplotlib.pyplot as plt
import random as r

# Data
L=[-2,2,3,-1,4,5,2,3,5,2,6]
# -2 o -1 är i bin -2 till 2 (exkl)
# 2,3,4,5,2,3,5,2 är i bin 2 till 6 exkl
#6 är i bin 6 till 10 exkl

# Kommandot för att plotta histogram

plt.hist(L, [-2,2,6,10,14,])

# De vanliga
plt.savefig("plot.png")
```

```
plt.show()
```



[32]:
```
import matplotlib.pyplot as plt
import random as r

# Data
L=[]
for i in range(1,1001,1):
    L.append(r.gauss(0,10))


# Kommandot för att plotta histogram
Bin=list(range(-30,31,1))
plt.hist(L, Bin)

# De vanliga
plt.savefig("plot.png")
plt.show()
```



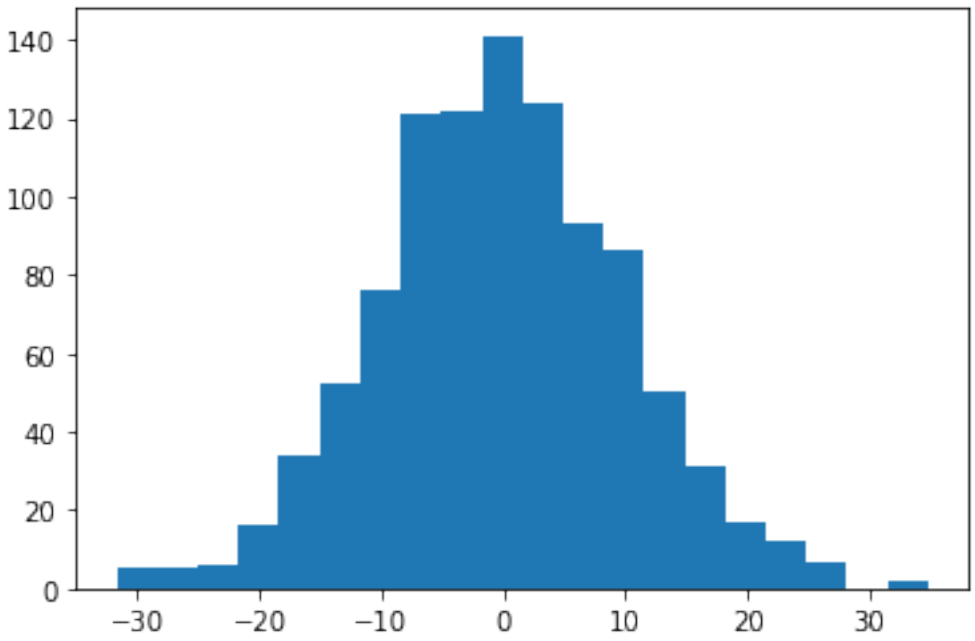
Kommentar angående `plt.hist(x, [0,2,4,6,8,10])`

Argumentet består av 2 listor. Datamängden ska finnas i listan x. Listan efter x är listan som anger var gränserna för de olika binen ska gå, de olika grupperna. Intervallet from. 0 till 2 är första binet; alltså inklusive 0 och exklusive 2. För det sista intervallet 8 till 10 gäller dock inklusive för båda gränserna. Vi ser från datamängden att det finns 5 stycken 1:or vilket ger en stapel med höjden 5 för intervallet 0 till 2; vi ser vidare att det finns 5 st 2:or och 2 st 3:or så 7 blir höjden för intervallet 2 till 4.

Vi kan också bara ange antalet intervall vi önskar och lämna resten till python som då konstruerar intervall som är lika stora. Anger vi inget intervall så är det förinställt 10 stycken.

[33]:
```
import matplotlib.pyplot as plt
import random as r

# Data
L=[]
for i in range(1,1001,1):
    L.append(r.gauss(0,10))


# Kommandot för att plotta histogram
#Bin=list(range(-30,31,1))
plt.hist(L, 20)

# De vanliga
plt.savefig("plot.png")
```

```
plt.show()
```



`plt.hist(x,6)` är ett anrop precis som när vi skriver en def, den returnerar värden i något som heter en tuple och den fungerar ungefär som en lista. Detta innebär att vi kan skriva ut värdena som används för att konstruera grafen numeriskt.

[36]:
```
import matplotlib.pyplot as plt

# Data
L = [1,1,1,2,2,2,4,4,4,5,5,5,5,5,6,6,7,7,7,8,8]

# [1,2[ 3 st 1
# [2,3[ 3 st 2
# [3,4[ 0 st 3
# Kommandot för att plotta histogram, nu med
# return-värdena så vi kan skriva ut dem
#n,bins,patches=plt.hist(L, 7)

Ut=plt.hist(L, 7)  # anrop funktion skickar 3 saker tillbak
print(Ut[0])  # antalet i varje bin
print(Ut[1])  # binens gränser


print("Antalet i varje intervall", n)
print("Binens gränser", bins)
# patches bryr vi oss inte om
```

```
# De vanliga
plt.savefig("plot.png")
plt.show()
```

```
[3. 3. 0. 3. 5. 2. 5.]
[1. 2. 3. 4. 5. 6. 7. 8.]
Antalet i varje intervall [3. 3. 0. 3. 5. 2. 5.]
Binens gränser [1. 2. 3. 4. 5. 6. 7. 8.]
```