Mer om funktioner och grafik i Python

1 Inledning

Först skall vi se lite på matriser och därefter på några funktioner som finns i det paket vi använder ihop med PYTHON, matematiska funktioner som sinus och cosinus samt funktioner för att bilda och operera på vektorer och matriser. Sedan ser vi lite på grafritning och annan grafik för att avslutningvis se hur man definerar egna funktioner.

2 Något om vektorer och matriser i `NumPy`

En matris är ett rektangulärt talschema

$$\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

Matrisen ovan har m rader och n kolonner, vi säger att den är av typ $m \times n$. Ett matriselement i rad nr i, kolonn nr j tecknas a_{ij}, där i är radindex och j är kolonnindex. Om vi använder `NumPy` i PYTHON skrivs detta `A[i-1,j-1]` och `shape(A)` ger matrisens typ.

En matris av typ $m \times 1$ kallas kolonnmatris (kolonnvektor) och en matris av typ $1 \times n$ kallas radmatris (radvektor):

$$\mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} c_1 & \dots & c_n \end{bmatrix}$$

Element nr i ges i Python av `b[i-1], c[i-1]` och antalet element ges av anropen `size(b), size(c)`. Som exempel tar vi

$$\mathbf{A} = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 0 & 2 & 4 & 6 & 8 \end{bmatrix}$$

Vi skriver in detta i Python enligt (Här markerar >>> att vi skriver raderna i Console).

```
>>> from numpy import *
>>> A=array([[1,4,7,10],[2,5,8,11],[3,6,9,12]])
>>> b=array([[1],[3],[5]])
>>> c=array([0,2,4,6,8])
```

I det här exemplet använder vi bara funktioner som finns i `NumPy` och inte i `Matplotlib` därför skriver vi `from numpy import *` allra först. Vi kan också skriva `from pylab import *` och importerar då funktioner både från `NumPy` och `Matplotlib`.

Vi använder fält, klassen `array` i `NumPy`, för att bygga matriserna. Varje rad i matrisen eller vektorn omges med hakparenteser, liksom hela matrisen.

Vill vi se vektorernas och matrisens värden kan vi skriva variabelnamnen. PYTHON markerar att det är fråga om fält (`array`) genom att infoga namnet `array` framför värdena.

```
>>> b
array([[1],
       [3],
       [5]])
>>> c
array([0, 2, 4, 6, 8])
>>> A
array([[ 1,  4,  7, 10],
       [ 2,  5,  8, 11],
       [ 3,  6,  9, 12]])
```

Indexeringen i vektorer och matriser i PYTHON börjar på `0`. T.ex. `b[0]` är första elementet i vektorn `b`, `b[1]` är andra elementet. Sista elementet i `b` får vi med `b[2]` eller `b[-1]`. Elementet på andra raden i 3:e kolonnen i `A` skrivs `A[1,2]`.

Uppgift 1 (a). Skriv in matriserna i PYTHON och skriv sedan ut matriselementen a_{23}, b_2, c_3. Prova `shape` och `size`. Ändra a_{23} till 15 genom att skriva `A[1,2]=15`.

Ibland vill vi se en tabell som en matris. Som exempel tar vi: Värmeförlusten hos den som vistas i kyla beror inte enbart på temperaturen, utan även på hur mycket det blåser. Tabellen visar vilken *effektiv temperatur* det blir vid olika temperaturer T (°C) och vindhastigheter v (m/s).

v \ T	10	6	0	-6	-10	-16	-26	-30	-36
2	9	5	-2	-9	-14	-21	-33	-37	-44
6	7	2	-5	-13	-18	-26	-38	-44	-51
10	6	1	-7	-15	-20	-28	-41	-47	-55
14	6	0	-8	-16	-22	-30	-44	-49	-57
18	5	-1	-9	-17	-23	-31	-45	-51	-59

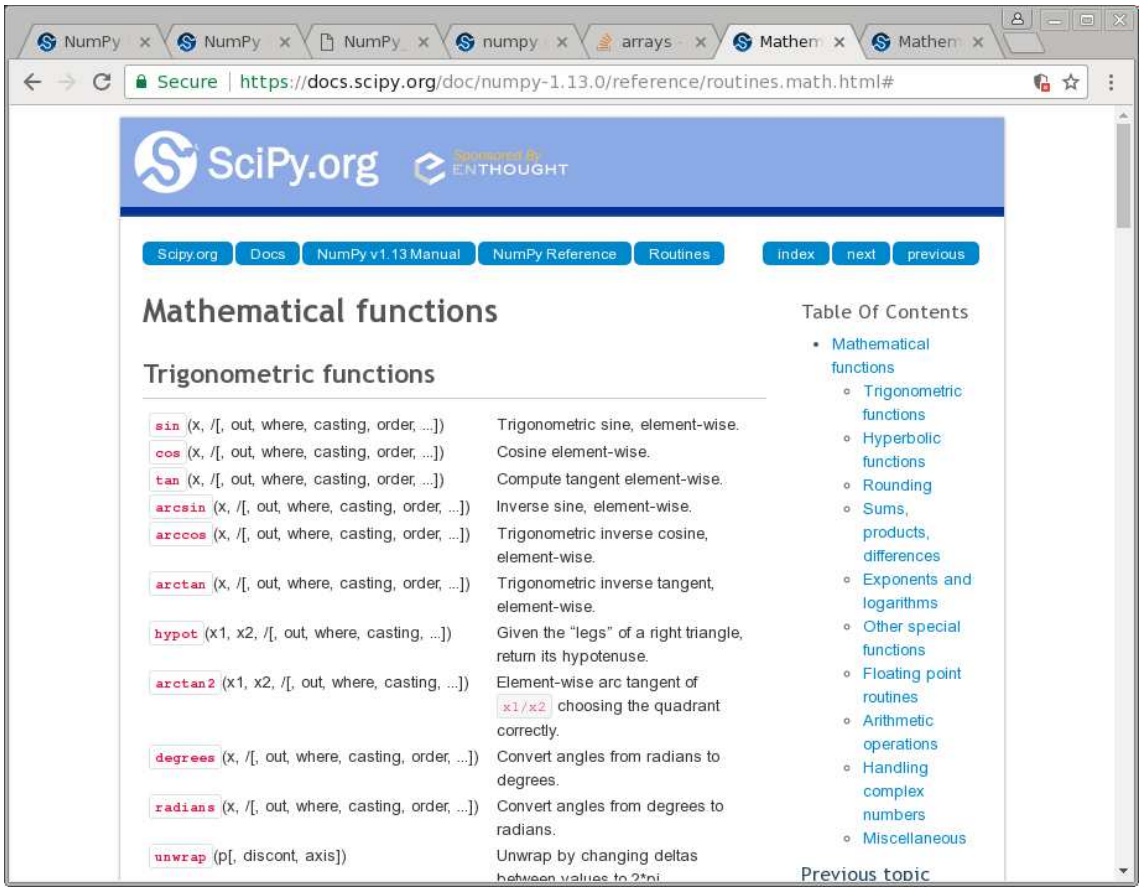
Om vi ville göra något med dessa data i PYTHON så skulle vi kunna lagra temperaturer och vindhastigheter i rad- eller kolonnvektorer och effektiva temperaturerna i en matris. (Vill du läsa mer om värmeförlust gå till SMHI:s hemsida och sök på vindavkylning.)

3 Funktioner i `NumPy`

3.1 Matematiska funktioner i `NumPy`

Vi tittar i dokumentationen för `NumPy` för att få en överblick över de matematiska funktioner som finns. Öppna en webbläsare och gå till

`https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.math.html`



Vi ser att funktionerna är grupperade, t.ex. en grupp med trigonometriska funktioner och en grupp med exponent- och logaritmfunktioner.

Funktioner som exempelvis sinus och cosinus, kan operera både på enskilda tal och på fält. Man får som resultat ett fält av samma storlek, vars element är funktionsvärdet av respektive element i argumentet.

Som exempel tar vi återigen

$$\mathbf{A} = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 0 & 2 & 4 & 6 & 8 \end{bmatrix}$$

som vi skriver in i PYTHON enligt

```
>>> from numpy import *
>>> A=array([ [1,4,7,10],[2,5,8,11],[3,6,9,12]])
>>> c=array([0,2,4,6,8])
```

Här markerar >>> att vi skrivit raderna i Console.

Nu beräknar vi sinus av vektorn $\mathbf{c}$ och matrisen $\mathbf{A}$ med

```
>>> v=sin(c)
>>> print(v)
[ 0.  0.90929743 -0.7568025  -0.2794155   0.98935825]
>>> V=sin(A)
```

```
>>> print(V)
[[ 0.84147098 -0.7568025   0.6569866  -0.54402111]
 [ 0.90929743 -0.95892427  0.98935825 -0.99999021]
 [ 0.14112001 -0.2794155   0.41211849 -0.53657292]]
```

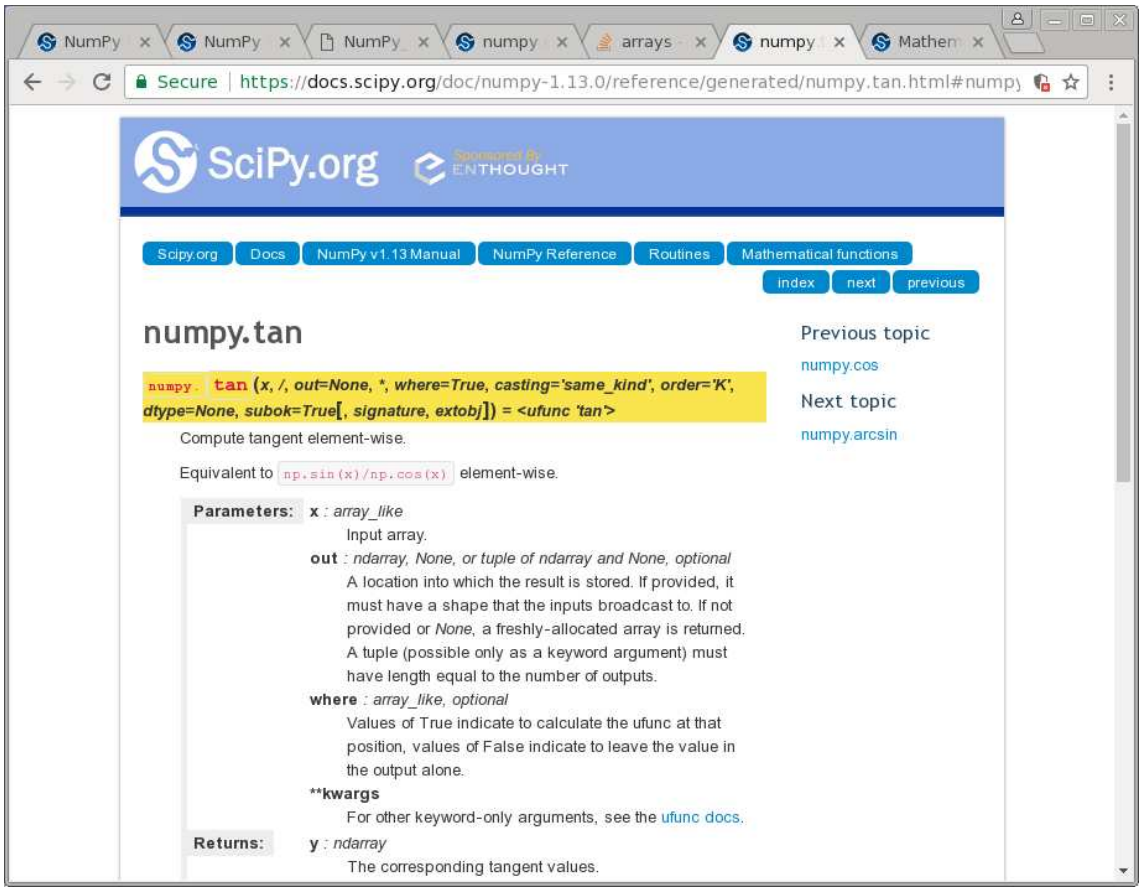
Man anger vektorn eller matrisen som argument till `sin` när man beräknar sinus för värdena. Funktionen beräknar sinus för alla värden i vektorn eller matrisen, elementvis.

När vi använder `print` för att skriva ut elementen i vektorn och matrisen skrivs inte ordet `array` framför fältet i utskriften.

Kommandonas hjälptexter är inte alls nybörjaranpassade. De beskriver bara hur kommandona är tänkta att användas. Dessutom är beskrivningen väldigt teknisk. Vi klickar på `tan`, dvs. tangensfunktionen, på dokumentationssidan och ser på hjälptexten.
Kommandot anropas med minst ett argument, ett tal, eller ett fält (vektor eller matris).

```
>>> a=tan(-pi)
>>> t=array([-pi,pi/2,pi])
>>> b=tan(t)
array([ 1.22464680e-16,  1.63312394e+16, -1.22464680e-16])
```

Kommandot är ekvivalent med att anropa `sin(x)/cos(x)`.



Uppgift 2. Rita upp tangensfunktionen på intervallet $0 \leq x \leq \pi$. Vad händer då $x = \dfrac{\pi}{2}$?

3.2 Några andra funktioner i NumPy

Vi har skrivit `from mumpy import *` för att få tillgång till de matematiska funktionerna och konstanterna i `NumPy`. I PYTHON kan det finnas flera funktioner med samma namn. Då finns funktionerna i olika paket. För att säkerställa att man anropar en funktion eller konstant från ett visst paket kan man namnge paketet när man importerar det. Det är vanligt att man ger `NumPy` namnet `np` och skriver

```
import numpy as np
```

Nu måste man infoga prefixet `np.` framför namnet så fort man använder någon funktion eller konstant från `NumPy`.

Vi använder samma vektor **c** och matris **A** från förra avsnittet och bildar (`>>>` betyder att vi skriver i Console)

```
>>> import numpy as np
>>> A=np.array([ [1,4,7,10],[2,5,8,11],[3,6,9,12]])
>>> c=np.array([0,2,4,6,8])
```

Vi har redan sett funktonerna `size` och `shape`. Antal element i vektorn **c** ges av

```
>>> l=np.size(c)
>>> l
5
```

Med `np.size` säkerställer vi att det är funktionen `size` från `NumPy` som avses. Antalet rader och kolonner **A** fås med

```
>>> [m,n]=np.shape(A)
>>> m
3
>>> n
4
```

Vi kan beräkna största och minsta element i fält med funktionerna `max` och `min`.

```
>>> v = np.max(c)
>>> v
8
>> v=np.max(A)
>>> v
12
```

Summan och produkten av elementen i vektorn fås med `sum` och `prod`.

```
>>> s=np.sum(c)
>>> s
20
>>> s=np.prod(A)
>>> s
479001600
```

Vill vi sortera en vektor i stigande ordning kan vi göra det med `sort`. Om vi skriver

```
>>> c=np.array([2,4,1,-9,0])
>>> sc=np.sort(c)
>>> print(sc)
[-9  0  1  2  4]
>>> print(c)
[ 2  4  1 -9  0]
```

blir `sc` en sorterad vektor och `c` är kvar som förut. För en matris blir det varje rad som sorteras om i stigande ordning.

```
>>> A=np.array([[2,4,1,-9,0],[-8,0,2,-3,5]])
>>> sA=np.sort(A)
>>> print(sA)
[[-9  0  1  2  4]
 [-8 -3  0  2  5]]
```

4 Hantering av vektorer och matriser i NumPy

4.1 Vektorer i NumPy

Vektorn $\mathbf{c} = (0, 2, 4, 6, 8)$ från förra avsnittet kan bildas på flera sätt i `NumPy`.

```
>>> from numpy import *
>>> c=array([0,2,4,6,8])
>>> c=arange(0,9,2)
>>> c
array([0, 2, 4, 6, 8])
```

Om man använder funktionen `arange` anger man `start, slut, steg`. Man får *från och med* `start` *till* `slut`. Man får inte till och med `slut`.

Med `start`=0, `slut`=8 och `steg`=2 får vi vektorn $(0, 2, 4, 6)$

```
>>> c=arange(0,8,2)
>>> c
array([0, 2, 4, 6])
```

För att få vektorn $(0, 2, 4, 6, 8)$ kan vi skriva

```
>>> c=arange(0,10,2)
>>> c
array([0, 2, 4, 6, 8])
```

Det sista talet blir det närmsta heltalet < 10 som passar. Vi kan också (som vi gjorde först) skriva

```
>>> c=arange(0,9,2)
>>> c
array([0, 2, 4, 6, 8])
```

Det sista talet blir heltalet närmst under 9 som passar.

Vi låter s få tredje värdet c_3 med `s=c[2]`. Det gäller att komma ihåg att index börjar med 0 i PYTHON.

```
>>> s=c[2]
>>> s
4
```

När man skapar nya vektorer eller delvektorer med hjälp av redan skapade vektorer måste man kopiera elementen till den nya vektorn. Vi kan bilda en ny vektor med samma värden som **c**

```
>>> kopia=c.copy()
>>> kopia
array([0, 2, 4, 6, 8])
```

Elementen i **c** *kopieras* till `kopia`.

Vi kan bilda vektorn **v** av andra och femte värdet, dvs. $\mathbf{v} = (c_2, c_5)$, med

```
>>> v=c[[1,4]].copy()
>>> v
array([2, 8])
```

Andra och 5:e elementet i **c** *kopieras* till **v**.

Vi kan bilda vektorn **v** av de tre första elementen i **v**, dvs. $\mathbf{v} = (c_1, c_2, c_3)$, med

```
>>> v=c[0:3].copy()
>>> v
array([0, 2, 4])
```

Man använder egentligen `start:slut:steg` när man indexerar i vektorn `c`. Om man utelämnar `steg` får man steget 1. Man får indexen *från och med* `start` till `slut`. Så `c[0:3]` ger elementen på platserna $0, 1$ och 2 i `c`.

Vi kan ändra ett element i **v**, t.ex. låta $v_2 = 0$, med

```
>>> v[1]=0
>>> v
array([2, 0])
```

4.2 Matriser i NumPy

Vi bildar samma matris **A** som i första avsnittet

```
>>> from numpy import *
>>> A=array([[1,4,7,10],[2,5,8,11],[3,6,9,12]])
>>> A
array([[ 1,  4,  7, 10],
       [ 2,  5,  8, 11],
       [ 3,  6,  9, 12]])
```

Vi låter s få värdet av elementet på rad 2, kolonn 3 i matrisen med `s=A[1,2]`.

```
>>> s=A[1,2]
>>> s
8
```

och vi bildar en ny radvektor $\mathbf{v}$ av rad 3, alla kolumner med

```
>>> v=A[2,:].copy()
>>> v
array([ 3,  6,  9, 12])
```

samt en vektor $\mathbf{u}$ av rad 2-3, kolonn 2 med

```
>>> u=A[1:3,1].copy()
>>> u
array([5, 6])
```

Vi använde `1:3` för att berätta vilka rader fältet som avsågs. Vi får raderna 1 och 2 i fältet, dvs. raderna 2 och 3 i matrisen. Kolonn 2 i matrisen har index 1 i fältet.

PYTHON gör automatiskt om kolonnvektorn vi indexerar till en radvektor.

Vi bildar en matris $\mathbf{V}$ av blocket rad 1-2, kolonn 2-3

```
>>> V=A[0:2,1:3].copy()
>>> V
array([[4, 7],
       [5, 8]])
```

4.3 Operatorer på vektorer och matriser i `NumPy`

Om vi har två vektorer $\mathbf{u} = (2, 3, 5)$ och $\mathbf{v} = (1, 2, 3)$ av samma typ och vill bilda summan $\mathbf{a} = \mathbf{u} + \mathbf{v}$ och skillnaden $\mathbf{b} = \mathbf{u} - \mathbf{v}$, så gör vi det med `a=u+v` respektive `b=u-v`. Operationerna sker elementvis

$$\mathbf{a} = \mathbf{u} + \mathbf{v} = (2, 3, 5) + (1, 2, 3) = (2 + 1, 3 + 2, 5 + 3) = (3, 5, 8)$$
$$\mathbf{b} = \mathbf{u} - \mathbf{v} = (2, 3, 5) - (1, 2, 3) = (2 - 1, 3 - 2, 5 - 3) = (1, 1, 2)$$

eller med andra ord $a_i = u_i + v_i$ och $b_i = u_i - v_i$.

T.ex. vid grafritning behövs de elementvisa motsvarigheterna till multiplikation och division

$$\mathbf{u} \texttt{ * } \mathbf{v} = (2, 3, 5) \texttt{ * } (1, 2, 3) = (2 \cdot 1, 3 \cdot 2, 5 \cdot 3) = (2, 6, 15)$$
$$\mathbf{u} \texttt{ / } \mathbf{v} = (2, 3, 5) \texttt{ / } (1, 2, 3) = (2/1, 3/2, 5/3) = (2, 1.5, 1.666\ldots)$$

Här har vi lånat beteckningar från PYTHON där vi skriver `u*v` respektive `u/v` för att utföra beräkningarna.

Vi behöver även elementvis upphöjt till, t.ex. kvadrering

$$\mathbf{u}\texttt{**}2 = (2, 3, 5) \texttt{ **}2 = (2^2, 3^2, 5^2) = (4, 9, 25)$$

Även här har vi lånat beteckningen ** från PYTHON.

Det finns en funktion `linspace` som är bra för att bygga upp vektorer och funktionerna `zeros` och `ones` för att bygga upp matriser fylla med nollor respektive ettor samt funktionen `eye` för att göra enhetsmatriser.

Uppgift 3. Använd `linspace`, läs hjälptexten, för att bilda vektorn $\mathbf{d} = (2, 5, 8, 11, 14)$. (Skriv `help(linspace)` i Console).

5 Grafik med `Matplotlib`

5.1 Grafritning

Vi har redan ritat några grafer av funktioner $f(x)$ över ett intervall $a \leq x \leq b$. Nu ser vi på ett exempel där vi behöver elementvisa operationer.

Exempel 1. Rita grafen till $f(x) = x\sin(x)$ över intervallet $0 \leq x \leq 8$.

Vi bildar en vektor $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ med värden jämnt fördelande över intervallet $0 \leq x \leq 8$. Sedan bildar vi vektorn

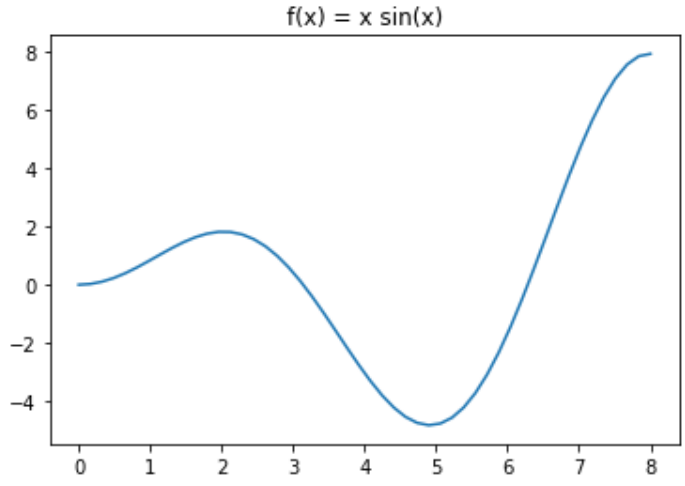
$$\mathbf{y} = (f(x_1), f(x_2), \cdots, f(x_n)) = (x_1\sin(x_1), x_2\sin(x_2), \cdots, x_n\sin(x_n)) = \mathbf{x} * \sin(\mathbf{x})$$

och ritar upp grafen. För att bilda vektorn $\mathbf{y}$ behövs den elementvisa multiplikationen.

Vi ritar grafen genom att skriva följander rader i Editorn.

```
from pylab import *
x=linspace(0,8)
y=x*sin(x)
plot(x,y)
title('f(x) = x sin(x)')
```

Vi använde funktioner både från paketet `NumPy` och `Matplotlib` därför skrev vi `from pylab import *` på första raden. Så här ser resultatet ut:



Uppgift 4. Rita grafen till $f(x) = x - x\cos(7x)$ över intervallet $0 \leq x \leq 8$.

5.2 Dela figuren

Ibland vill man ha flera koordinatsystem i samma figur, då kan man använda funktionen `subplot` från `Matplotlib`.

Exempel 2. Vi skall i samma figur göra tre olika koordinatsystem. I dessa skall vi rita grafen av $\sin(x)$, $\cos(x)$ respektive $\tan(x)$ över intervallet $-\frac{\pi}{2} \leq x \leq \frac{\pi}{2}$.

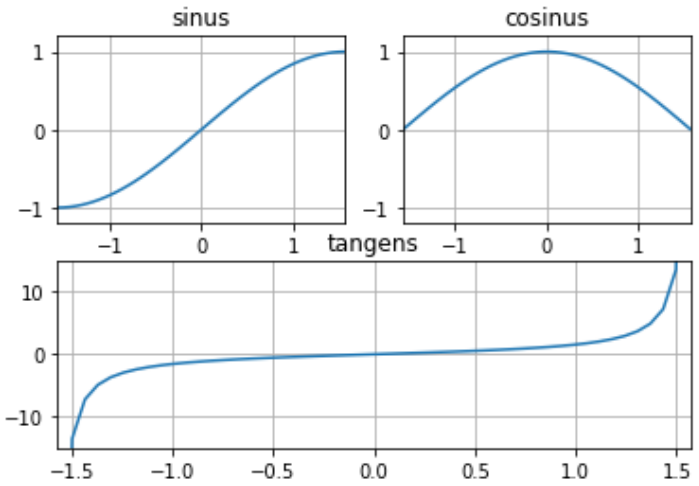
```
# -*- coding: utf-8 -*-
from pylab import *
s=0.01
x=linspace(-pi/2+s,pi/2-s)
subplot(221)           # delar in Figuren i 2x2 delar och gör 1:a aktiv
plot(x,sin(x))
axis([-pi/2, pi/2, -1.2, 1.2])
grid()
title('sinus')

subplot(222)           # delar in Figuren i 2x2 delar och gör 2:a aktiv
plot(x,cos(x))
axis([-pi/2, pi/2, -1.2, 1.2])
grid()
title('cosinus')

subplot(212)           #  delar in Figuren i 2x1 delar och gör 2:a aktiv
plot(x,tan(x))
axis([-pi/2, pi/2, -15, 15])
grid()
title('tangens')
```

(Allra först i koden står raden `# -*- coding: utf-8 -*-`. Ibland får man problem om man använder de svenska bokstäverna åääöÅÄÖ i Pythonscript. De tre kommentarerna innehåller alla ordet "gör").

Så här ser figuren ut:



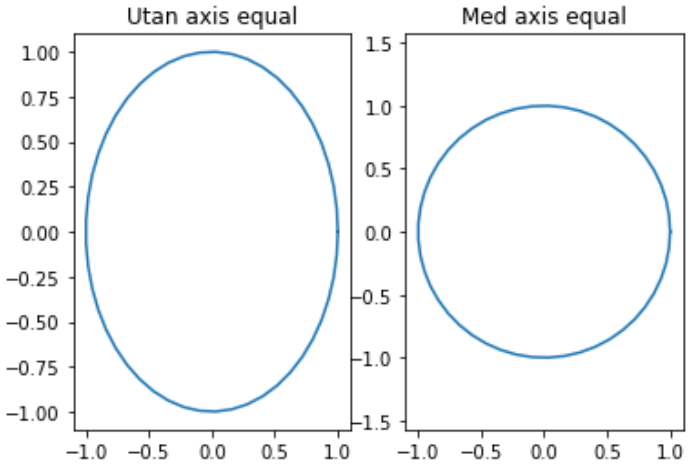
5.3 Kurvritning

Nu skall vi rita s.k. parameterframställda kurvor. Som exempel tar vi enhetscirkeln

$$t \mapsto (x(t), y(t)) = (\cos(t), \sin(t)), \quad 0 \leq t \leq 2\pi$$

När man ritar sådana kurvor ritar man inte ut parametern t utan enbart x- och y-värdena.

```
from pylab import *
t=linspace(0,2*pi)
x=cos(t); y=sin(t)
subplot(121)
plot(x,y)
title('Utan axis equal')
```



För att cirkeln inte skall se ut som en oval måste vi använda `axis` och ange `equal`.

```
subplot(122);
plot(x,y)
axis('equal')
title('Med axis equal')
```

Uppgift 5. Rita kurvorna $t \mapsto (x(t), y(t)) = (\cos(t) + \cos(3t), \sin(2t))$ och $t \mapsto (x(t), y(t)) = (\cos(t) + \cos(4t), \sin(2t))$, för $0 \leq t \leq 2\pi$. Använd `plot` och `subplot`.

5.4 Polygontåg

Ett polygontåg som ges av $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$, ritas upp i PYTHON genom att man bildar vektorerna $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ och $\mathbf{y} = (y_1, y_2, \cdots, y_n)$ och sedan gör `plot(x,y)`.

Grafritning är ett polygontåg vi ritar upp. Tag t.ex. grafen till $f(x) = \sin(x)$ för $0 \leq x \leq 2\pi$. Vi har då $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ med $0 = x_1 < x_2 < \cdots < x_n = 2\pi$ och $\mathbf{y} = (y_1, y_2, \cdots, y_n)$ med $y_i = \sin(x_i)$. Sedan ritar vi upp med `plot(x,y)`.

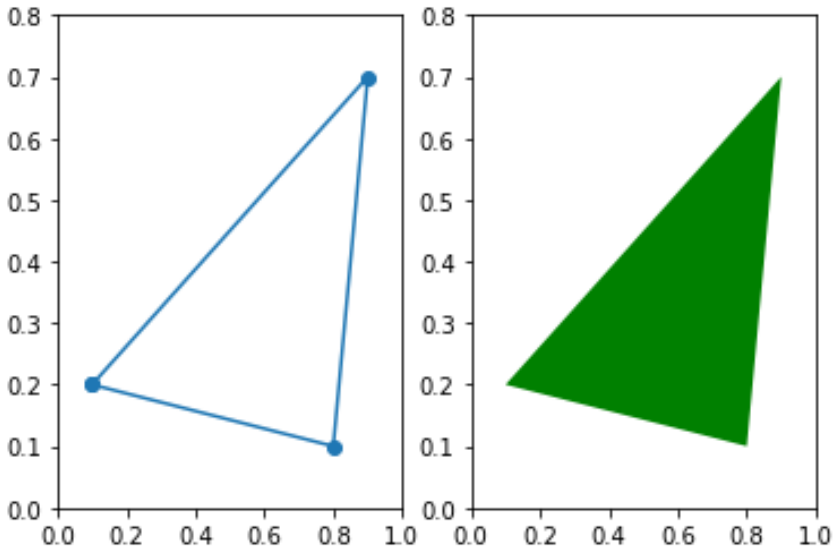
Om polygontåget är slutet, dvs. $x_n = x_1$ och $y_n = y_1$, och om det inte korsar sig självt så omsluter det ett område i planet, ett s.k. polygonområde. Vi kan använda `fill` för att färglägga ett sådant område.

Vi ritar upp polygontåg som ges av $(0.1, 0.2), (0.8, 0.1), (0.9, 0.7), (0.1, 0.2)$, dvs. en triangel.

```
from pylab import *
x=array([0.1, 0.8, 0.9, 0.1 ])
y=array([0.2, 0.1, 0.7, 0.2 ])
subplot(121)
plot(x,y,'-o')
axis([0,1,0,0.8])
```

Med `'-o'` anger vi att punkterna både skall förbindas med räta linjer och markeras med små ringar. Vi fyller området med grön färg och vi använder `axis` för att få lite "luft" runt triangeln.

```
subplot(122)
fill(x,y,'green')
axis([0,1,0,0.8])
```



Uppgift 6. Rita en cirkel fylld med grön färg, rita sedan en kvadrat inskriven i cirkeln och fyll kvadraten med gul färg.

6 Egna funktioner

Exempel 3. Vi skall rita grafen av

$$f(x) = \frac{\sin(ax)}{x}$$

över intervallet $-5 \leq x \leq 5$, för $a = 1, 2,$ och 3.

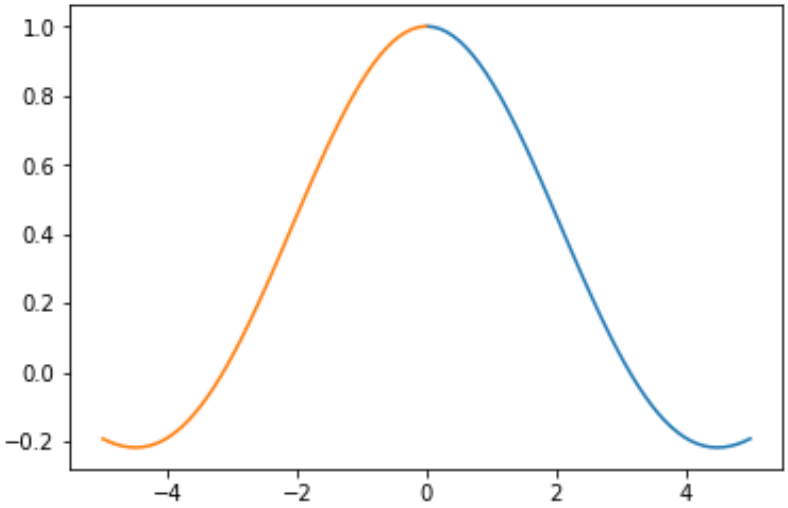
Nu behöver vi elementvis division och vi inför en funktion. Vi ser först på fallet $a = 1$

```
from pylab import *
def f(x):
    return sin(x)/x
```

På tredje raden står först fyra blanksteg, sedan `return sin(x)/x`.

Vi måste tänka på att $f(x)$ inte definierad i $x = 0$. Finns gränsvärdet och vad blir det i så fall?

```
s=0.01                # s för att separera nollan
xn=linspace(-5,-s)    # xn negativa x-värden
xp=linspace(s,5)      # xp positiva x-värden
plot(xp,f(xp))
plot(xn,f(xn))
```



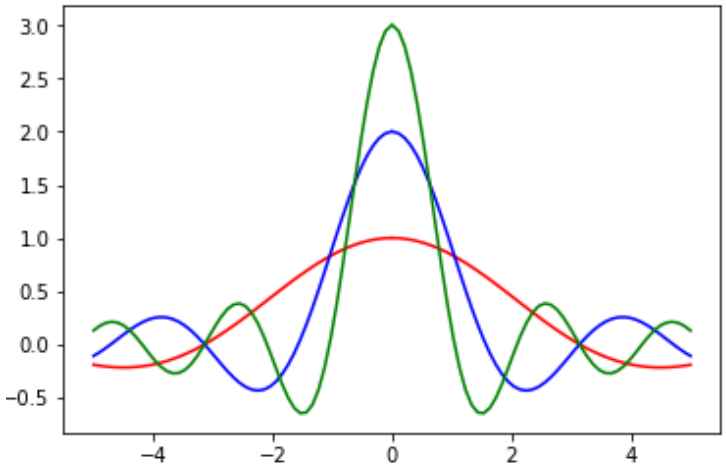
Nu skall vi ta fallen $a = 2$ och 3 också. Enklast är om vi gör om vår funktion så att även a blir ett argument. Vi öppnar en editor och skriver in koden.

```
# -*- coding: utf-8 -*-
from pylab import *
def f(x,a):
    return sin(a*x)/x

s=0.01                # s för att separera nollan
xn=linspace(-5,-s)    # xn negativa x-värden
xp=linspace(s,5)      # xp positiva x-värden

plot(xn,f(xn,1),'red',xp,f(xp,1),'red')      # a=1
plot(xn,f(xn,2),'blue',xp,f(xp,2),'blue')    # a=2
plot(xn,f(xn,3),'green',xp,f(xp,3),'green') # a=3
```

Vi kör programmet från ett terminalfönster och får följande figur.



I figuren ser det ut som $f(x) \to a$ då $x \to 0$. Tänk dock på att en figur inte är något bevis!
Vi definerade en funktion enligt

```
def funktionsnamn(parametrar):
    satser
    return svar
```

Här är `funktionsnamn` namnet på funktionen. Med `parametrar` avser vi indata till funktionen (argumenten), ofta en variabel, ibland flera. Andra raden `satser` kallas funktionskroppen. Här står koden som ska köras när funktionen anropas. Man måste skriva satserna en bit in på raden, ofta använder man fyra blanktecken. På sista raden står `return svar`. Det värde som finns i `svar` skickas tillbaka från funktionen när den anropas.

Exempel 4. En kastbana utan luftmotstånd beskrivs av

$$y(x) = y_0 - \frac{g}{2v_0^2\cos^2(\theta)}\left(x - \frac{v_0^2\sin(2\theta)}{2g}\right)^2 + \frac{v_0^2\sin^2(\theta)}{2g}$$

där v_0 är utkastfarten, y_0 är utkasthöjden, θ är utkastvinkeln och g är tyngdaccelerationen.

Först gör vi en function med namnet `kastbana` som beskriver kastbanan för olika utkastvinklar.

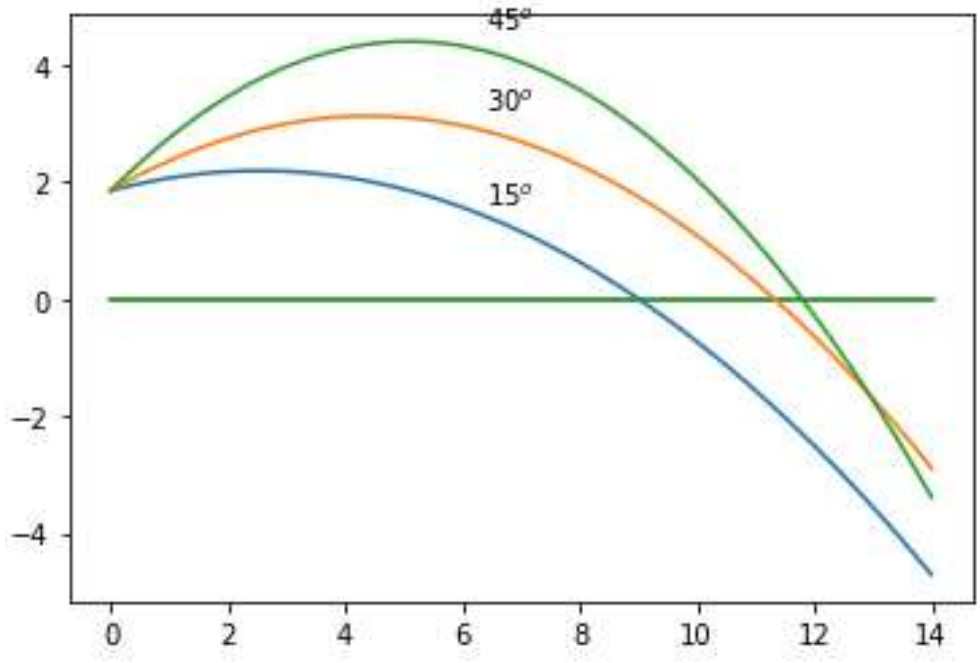
```
def kastbana(x,theta):
    t=theta*pi/180;     # theta i grader, t i radianer
    v0=10; y0=1.85; g=9.81
    a=g/(2*v0**2*cos(t)**2)
    b=v0**2*sin(2*t)/(2*g)
    c=v0**2*sin(t)**2/(2*g)
    y=y0-a*(x-b)**2+c
    return y
```

Vi fortsätter sedan på vårt script och vi tar $v_0 = 10$ m/s, $y_0 = 1.85$ m och ritar kastbanorna för några olika utkastvinklar.

```
x=linspace(0,14)
plot([0 14],[0 0],'green')  # Grön gräsmatta

plot(x,kastbana(x,15)), text(6.4,1.6,'$15^o$')
plot(x,kastbana(x,30)), text(6.4,3.2,'$30^o$')
plot(x,kastbana(x,45)), text(6.4,4.6,'$45^o$')
```

Så här ser det ut när vi ritat graferna. Vi har också placerat ut lite förklarande text vid graferna med funktionen `text`. Dollartecknen (`$`) som omger gradtalen i anropet till `text` markerar för funktionen att texten ska tolkas som en matematisk formel. Det innebär bl.a. att tecknet som kommer efter `^` upphöjs i utskriften.



Uppgift 7. Skriv den function och det script för kastbanan som vi pratar om i exemplet. Rita graferna. Varför delar vi upp funktionsuttrycket för $y(x)$ i flera delar?

7 Vidare läsning

Programspråket PYTHON är ganska omfattande och det finns väldigt mycket skrivet både om själva språket och alla paket som finns att importera. Mycket av informationen finns på nätet.

Vi har använt `array` från paketet `NumPy` för att representera vektorer och matriser. I andra avsnittet skapade vi några enkla fält för att representera en radvektor, en kolonnvektor och en matris. Vi använde funktionerna `size` och `shape` för att avgöra antalet element respektive matristyp. I avsnitt 3.1 använde vi några enkla matematiska funktioner, `sin`, `cos` och `tan`, även dessa från paketet `NumPy`. Funktionerna `max`, `min`, `sort`, `sum`, `prod` som användes i avsnitt 3.2 finns också i paketet `NumPy`. I det fjärde avsnittet tittade vi mer i detalj på hur man skapar fält med `array` och hur man indexererar i dem för att skapa nya delfält och för att ändra elementvärden i dem.

Organisationen SciPy (Scientific Computing Tools for Python) har en mycket omfattande beskrivning av paketet `NumPy` och dess innehåll. Framställningen är inte helt nybörjarvänlig, men den innehåller gott om exempel, så en hel del av texten förstår man även som nybörjare. Framförallt kan man använda sidan och slå upp detaljer i. Organisationen SciPy har hemsida

`https://www.scipy.org/`

Klicka på länken `NumPy` som tar dig till webbsidan `www.numpy.org`. Välj t.ex. länken `NumPy Tutorial`.

Om du även valt att titta på språket MATLAB finns en sida som jämför språken PYTHON och MATLAB. Välj länken `NumPy for MATLAB© Users` på webbsidan `www.numpy.org`.

I avsnitt 5 använde vi funktioner från paketet `Matplotlib` för att rita och administrera figurer. Organisationen SciPy (Scientific Computing Tools for Python) har även här en omfattande beskrivning av paketet och dess innehåll. Gå till `https://www.scipy.org/` och välj `Matplotlib`. Klicka sedan på t.ex. `examples`, `tutorials` eller `pyplot`. Under länkarna `examples` och `tutorials` finns fullt med exempel, rätt många går att följa även om de inte är nybörjaranpassade. Det gäller att sovra, läsa det man förstår och behöver, och strunta i resten så länge. Man kan få en del tips och kanske hämta inspiration. Länken `pyplot` (på sidan `https://matplotlib.org/`) innehåller en lista med alla funktioner som finns i `pyplot`.