Laboration 1, M0043M, VT16

Senaste inlämning för den skriftliga redogörelsen är för

- Laboration 1: 18 april 2016,
- Laboration 2: 9 maj 2016.

Observera Samtliga laborationer skall vara godkända senast 23 maj 2016. Eventuella kvarvarande laborationer/returer efter detta datum underkänns och laborationerna måste göras om vid nästkommande kurstillfälle HT 2016.

Introduktion till Python–Teoridel

1 Inledning

Vi skall använda programspråket Python i våra laborationer. Python är ett interpreterande högnivåspråk, där program, skrivna i Python interpreteras (tolkas) av pythontolken. Språket skapades 1990 av den nederländske programmeraren Guido van Rossum.

Python är ett kraftfullt verktyg, som erbjuder såväl interaktiv exekvering via kommandon, som exekvering via s.k. skript (dvs kommandon på fil). Språket är skrivet i öppen källkod, vilket betyder att Python är gratis. Python är portabelt och körs i de vanligaste miljöerna, t ex Mac, Linux, Windows,...

Python är inte stort i sig, men kan bli enormt omfattande eftersom man kan importera diverse moduler, skrivna för vitt skilda tillämpningar.

När man installerar Python, följer ungefär 350 moduler med i installationen. övriga moduler finns för nedladdning. Eller också kan man skriva egna moduler, allt efter behov.

2 Räkning med tal

Python kan användas som en avancerad miniräknare. För att multiplicera talen 12 och 13 skriver vi

```
>>> 12*13
```

Resultatet skrivs ut på skärmen

```
>>> 156
```

Upphöjt till markeras med

**

För att beräkna 3^4 skriver vi

```
>>> 3**4
```

varvid Python returnerar

```
>>> 81
```

Normalt placerar man resultat i egendefinierade variabler.

```
>>> from pylab import *  # Vi importerar allt fran pylab-modulen
>>> x=pi
>>> y=x/2
>>> print x,y
3.14159265359 1.57079632679
```

Första kommandot tilldelar x värdet π medan andra kommandot tilldelar y värdet $x/2 = \pi/2 \approx 1.5708$.

Ett stort antal funktioner är definierade i Python, se avsnitt 3.5. Om vi till ovanstående kommandon tillfogar

```
>>> z=sin(y)
```

returnerar Python

```
>>> print z
1.0
```

Variabeln z har tilldelats värdet $\sin(y)$ vilket är lika med 1 då $y = \pi/2$.

3 Vektorer

Antag att vi vill plotta funktionen $y = f(x) = \sin(2\pi x)$ i intervallet $[0, 1]$. För att göra detta måste man:

1. definiera ett antal x-värden i intervallet

$$0 = x_1 < x_2 < \ldots < x_n = 1$$

2. beräkna funktionsvärdet för varje x-värde

$$y_k = f(x_k), \quad k = 1, \ldots, n.$$

3. rita polygonlinjen som förbinder punkterna $(x_1, y_1), \ldots, (x_n, y_n)$.

Ovanstående exempel visar att det är viktigt att kunna generera följder av x- och y-värden. Vi börjar med att diskutera hur detta kan göras i Python.

3.1 Generering av vektorer

En vektor x är en samling av reella tal

$$x = (x_1, x_2, \ldots, x_n).$$

I Python är längden av vektorn är lika med antalet element. För att generera vektorn

$$\mathbf{v} = \begin{bmatrix} 1.1 \\ 2.2 \\ 3.3 \end{bmatrix}$$

och bestämma antalet element i vektorn skriver man

```
>>> v=[1.1 2.2 3.3]

>>> print v
[1.1, 2.2, 3.3]


>>> len(v)
3
```

Observera att `len(v)` inte är lika med $\|v\|$.

De tre elementen är nu lagrade i vektorn med namnet `v`.

Observera att elementindex startar från 0. För att inspektera exempelvis element två skriver man

```
>>> print v[1]
```

Python svarar då

```
2.2
```

vilket är värdet av elementet. Om vi vill ändra värdet av element två till ett nytt värde, säg 4.2, skriver vi

```
>>> v[1]=4.2
```

varvid Python returnerar den nya vektorn

```
>>> print vektor
[1.1, 4.2, 3.3]
```

Som vi upptäcker, genererar Python inga svarsutskrifter om man inte begär det. Till exempel genererar kommandot

```
y=[pi, 0, 3];
```

en vektor $y = (\pi, 0, 3)$ utan att resultatet visas på skärmen.

3.2 Arange – alternativ vektorgenerering

Det finns alternativa kommandon för att generera vektorer. Kommandot

```
>>> from numpy import *  # Vi importerar allt fran modulen numpy
>>> v2=arange(1,11,3)
```

genererar vektorn

```
>>> print v2
[ 1  4  7 10]
```

Avståndet mellan elementen i vektorn kallas *steglängden*. Vektorn ovan har steglängd tre. För att generera en vektor med steglängd 0.5 skriver man

```
>>> v3=arange(20,22.5,0.5)
```

vilket genererar vektorn

```
>>> print v3
[ 20.   20.5  21.   21.5  22. ]
```

Den allmänna formen av kolon-notationen är

```
>>> x=arange(a,b,h)
```

där a är startvärdet, h steglängden och b slutvärdet. Notera att b **inte** inkluderas i vektorn. Notera att även negativa steglängder är tillåtna. Till exempel ger

```
>>> v4=arange(10,-2,-2)
```

resultatet

```
>>> print v4
[10  8  6  4  2  0]
```

3.3 Linspace

Med kommandot `linspace` genereras vektorer med ett bestämt antal element.

Kommandot

```
>>> x=linspace(a,b,num=n)
```

ger en vektor med n element jämnt fördelade i intervallet $[a, b]$. Till exempel ger

```
>>> v5=linspace(0,1,num=11)
```

vektorn

```
print v5
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1. ]
```

3.4 Aritmetiska operationer på vektorer

Antag att vi har en vektor $\mathbf{a} = \begin{bmatrix} -1 \\ 0 \\ 2 \end{bmatrix}$. Kommandona

```
>>> a=array([-1,0,2])
>>> b=3*a
>>> c=a+2
print b,c
[-3  0  6] [1 2 4]
```

ger $\mathbf{b} = \begin{bmatrix} -3 \\ 0 \\ 6 \end{bmatrix}$ och $\mathbf{c} = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$. I första fallet har alla elementen i $\mathbf{a}$ har multiplicerats med tre, i andra fallet har talet två adderats till alla elementen i $\mathbf{a}$.

Antag nu att vi har två vektorer som innehåller lika många element, t. ex. $\mathbf{s} = \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix}$ och $\mathbf{t} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$.

Kommandona

```
>>> t=array([2.0,3.0,4.0])
>>> s=array([-1.0,1.0,2.0])
>>> u=s+t
>>> v=s*t
>>> w=s/t
>>> z=t**2
>>> print u,v,w,z
[ 1.  4.  6.] [-2.  3.  8.] [-0.5  0.33333333  0.5 ] [ 4.  9.  16.]
```

ger $\mathbf{u} = \begin{bmatrix} 1 \\ 4 \\ 6 \end{bmatrix}$, $\mathbf{v} = \begin{bmatrix} -2 \\ 3 \\ 8 \end{bmatrix}$, $\mathbf{w} = \begin{bmatrix} -1/2 \\ 1/3 \\ 1/2 \end{bmatrix}$ och $\mathbf{z} = \begin{bmatrix} 4 \\ 9 \\ 16 \end{bmatrix}$.

I första fallet har vektorerna $\mathbf{s}$ och $\mathbf{t}$ adderats elementvis. I andra och tredje fallet har vektorerna multiplicerats respektive dividerats elementvis. I det sista fallet har samtliga element i $\mathbf{t}$ kvadrerats.

Man skall vara medveten om att det inte går att addera, multiplicera eller dividera vektorer med olika längd.

Provkör ovanstående exempel på egen hand som nyttig övning. Prova med följande alternativa vektorgenerering innan du kör vektorkalkylen:

```
>>> t=array([2,3,4])
>>> s=array([-1,1,2])
```

Förklara resultatet.

3.4.1 Skalär- och vektorprodukt, längd

Python har ett antal inbyggda funktioner, knutna till vektorer. Vi ska titta närmare på några av dem i nästkommande laboration. Låt oss exemplifiera.

- Funktionen `dot(u,v)` returnerar skalärprodukten $\mathbf{u} \bullet \mathbf{v}$,
- Funktionen `cross(u,v)` returnerar vektorprodukten $\mathbf{u} \times \mathbf{v}$,
- Funktionen `norm(u)` returnerar vektorlängden $\|\mathbf{u}\|$.

Vi provar dessa funktioner på vektorerna **s** och **t** ovan. Resultatet blir som förväntat:

```
>>> from pylab import *  # Vi importerar allt fran pylab-modulen
>>> ip=dot(s,t)
>>> langd=norm(s)
>>> kryss=cross(s,t)
>>> print ip,langd,kryss
9.0 2.44948974278 [-2.  8. -5.]
```

Kontrollera resultatet med handräkning.

3.4.2 Projektion

Exempelvis kan vi beräkna projektionen av vektorn **t** på vektorn **s** med projektionsformeln.

```
>>> e=s/norm(s)
>>> tproj=dot(t,e)*e
>>> print tproj

[-1.5  1.5  3. ]
```

Kontrollera även här resultatet med handräkning.

3.5 Funktionsevaluering

Python har ett stort antal inbyggda matematiska funktioner. En del av dessa funktioner är listade nedan.

`abs(x)`	ger absolutbeloppet av x, dvs $\lvert x \rvert$.
`sqrt(x)`	ger kvadratroten av x, dvs $\sqrt{x}$.
`exp(x)`	ger exponentialfunktionen av x, dvs e^x.
`log(x)`	ger naturliga logaritmen av x, dvs $\ln x$.
`log10(x)`	ger 10-logaritmen av x, dvs $\lg x$.
`sin(x)`	ger $\sin x$ där x är i radianer.
`cos(x)`	ger $\cos x$ där x är i radianer.
`tan(x)`	ger $\tan x$ där x är i radianer.
`asin(x)`	ger $\arcsin x$, dvs $\sin^{-1} x$.
`acos(x)`	ger $\arccos x$, dvs $\cos^{-1} x$.
`atan(x)`	ger $\arctan x$, dvs $\tan^{-1} x$.

Funktionerna i Python accepterar förutom tal även vektorer som argument. Om vi till exempel definierar en vektor

```
>>> x=[1,11,1]
```

och skriver

```
>>>  y=sqrt(x)
```

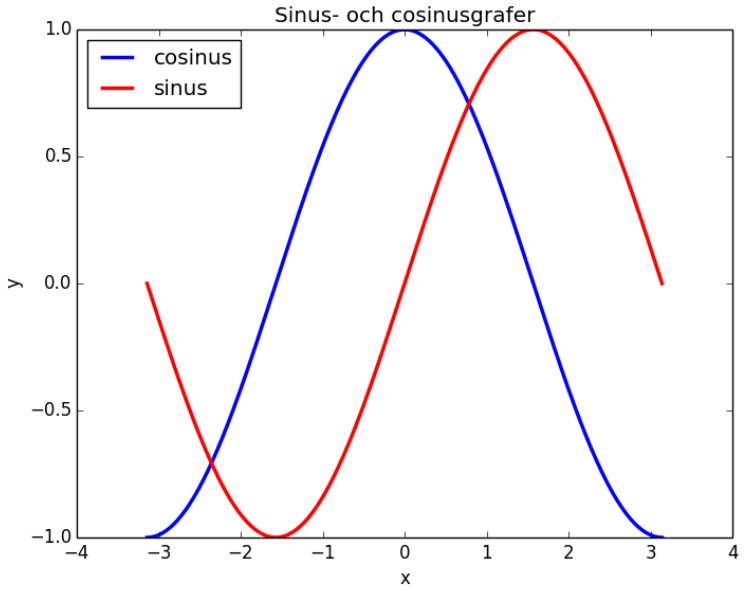
får vi en vektor y vars element består av kvadratrötterna av heltalen från ett till tio.

4 Plottning

För att plotta funktionerna $y = \sin x$ och $y = \cos x$ i intervallet $-\pi \leq x \leq \pi$ med heldragna linjer med olika färg skriver man

Exempel

```
>>> from pylab import *  # Vi importerar allt fran pylab-modulen
>>> X = linspace(-pi, pi, 300)
>>> C, S = cos(X), sin(X)
>>> plot(X, C, color="blue", linewidth=2.5, linestyle="-", label="
                                        cosinus")
>>>  plot(X, S, color="red",  linewidth=2.5, linestyle="-", label="sinus
                                        ")
>>> legend(loc='upper left')
>>> axis(-4,4,-1,1)
>>> xlabel('x')
>>> ylabel('y')
>>> title('Sinus- och cosinusgrafer')
>>> show()
```



Försök förstå de olika kommandona i ovanstående exempel.

Alternativt kan man använda följande teknik för att välja vilken linjetyp, punkttyp eller färg man önskar på kurvan. Den allmänna formen av detta plotkommando är

```
plot(x,y,'str')
```

där `str` är en teckensträng som talar om vilken linjetyp, punkttyp eller färg man önskar på kurvan. Några tillval är listade i nedanstående tabell.

Punkttyper		**Linjetyper**	
`'.'`	punkt	`'-'`	heldragen linje
`'o'`	fyllda ringar	`'--'`	streckad linje
`'^'`	fylld triangel	`'-.'`	punkt-streckad linje
		`':'`	prickad linje
		Färgtyper	
		`'g'`	grön
		`'m'`	magenta
		`'b'`	blå
		`'c'`	cyan
		`'k'`	svart
		`'y'`	gul
		`'r'`	röd

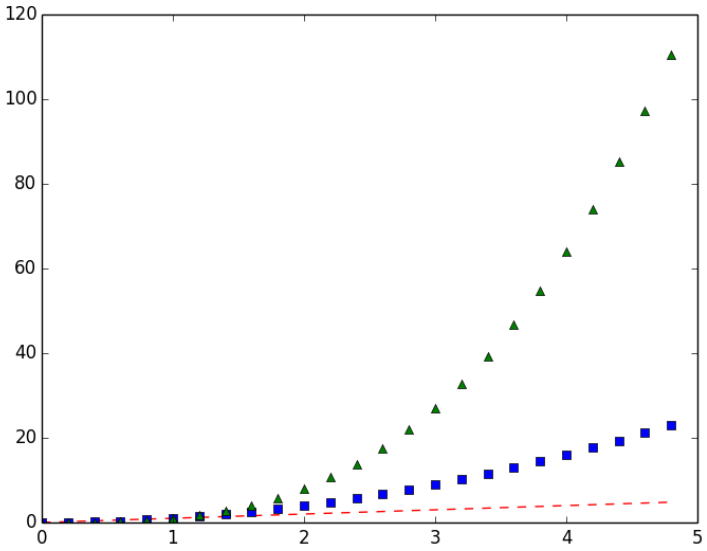
För fler tillval, använd hjälpfunktionen i Object Inspector.

Förutom att kontrollera punkt- linje- och färgtyp enligt ovan kan man också, enligt tidigare exempel, skriva text på axlar och i graffönstret.

Följande två exempel visas hur man kan använda en del av ovanstående kommandon. Försök följa och förstå de olika kommandona.

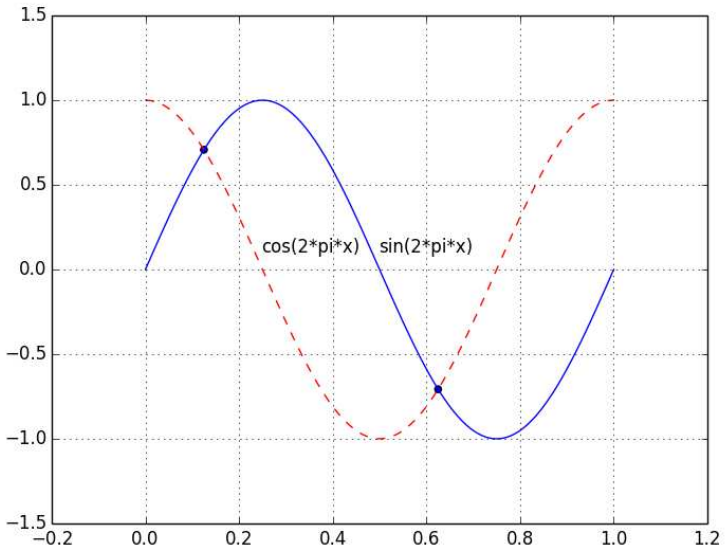
Exempel

```
>>> t=arange(0., 5., 0.2)
# rod streckad, bla kvadrat, gron triangel
>>> plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
>>> show()
```



Exempel

```
>>> x=linspace(0.,1.,100)
>>> y1,y2=sin(2*pi*x),cos(2*pi*x)
>>> plot(x,y1,'b-',x,y2,'r--')
>>> text(0.5,0.1,'sin(2*pi*x)')
>>> text(0.25,0.1,'cos(2*pi*x)')
>>> scatter(0.625,-1/sqrt(2))  #Genererar punkt i en given koordinat
>>> scatter(0.125,1/sqrt(2))
>>> grid(True)    # Rutnat
>>> show()
```



5 Scriptfiler och funktionsfiler

Det är ofta bekvämt att skriva kommandon i en *scriptfil*. Genom att skriva namnet på scriptfilen vid kommandopromtern kommer kommandona i filen att utföras i ordning. I Python har scriptfiler alltid suffixet (extension) `.py`. M filer skapas i tre steg:

1. Gå in i Spyders *Editor*.
2. Skriv in kommandona i editorn.
3. Spara filen. Observera att namn på scriptfiler ej får börja med siffror. Man ska också se till att man inte ger sin fil ett fördefinierat namn.

Kör scriptfilen med att trycka F5 (Run file).

5.1 Funktionsfiler

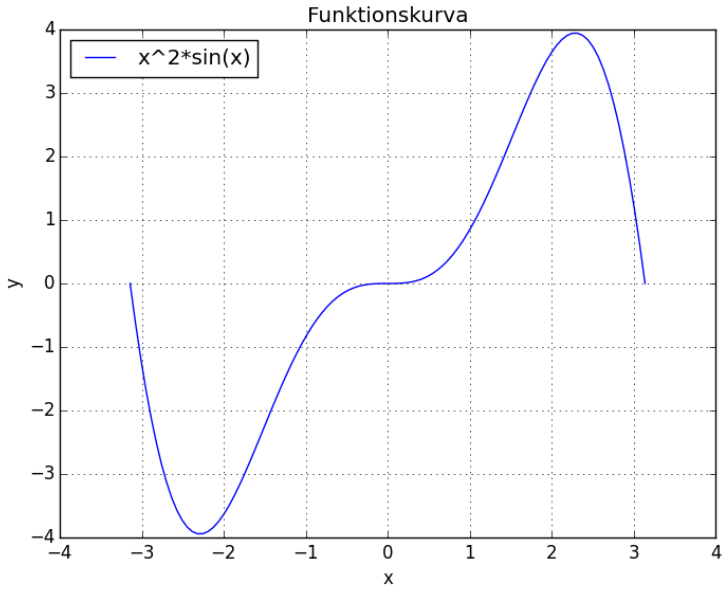
Funktionsfiler är en speciell typ av scriptfiler som definierar en funktion av en eller flera variabler. Som ett exempel kan vi ta funktionen $f(x) = x^2 \sin(x)$. Om vi refererar till funktionen som `fun1` så ges denna av funktionsfilen

```
def fun1(x):
    return x**2*sin(x)
```

Följande script-/funktionsfil ritar kurvan över funktionen `fun1` i intervallet$[-\pi, \pi]$.

Exempel

```
from pylab import *  # Vi importerar allt fran pylab-modulen
# Funktionen definieras
def fun1(x):
    return x**2*sin(x)
# Har borjar scriptet
x=linspace(-pi,pi,100)
y=fun1(x)  #Anrop av fun1
plot(x,y,'b-', label="x^2*sin(x)")
xlabel('x')
ylabel('y')
legend(loc='upper left')
grid(True)
title('Funktionskurva')
show()
```



6 Förberedelseuppgifter

1. På internet finns instruktiva filmer, t.ex. `http://www.youtube.com/watch?v=A6_gh0vrZ-E`.

2. Läs igenom laborationens teoridel, avsnitt 1-5. Kör teoridelens exempel.

7 Uppgiftsdel

Anvisningar

- Följ anvisningarna i ”Labb-PM, VT16”, som du kan ladda ner från Canvas.
- Lämna in en så enkel rapport som möljigt, utan – **detta är viktigt** – att utelämna python-kod, plottar och körningsresultat.
- Rapporten ska vara ett pdf-dokument (Konvertera till pdf från lämpligt ordbehandlingsprogram).
- OBS Viktigt Glöm inte namn på gruppmedlemmar och gärna epostadresser.

  Namnge dokumentet så att identifiering lätt kan ske.
- De obligatoriska uppgifterna ska göras i form av scriptfiler, enligt de beskrivna metoderna i detta dokument.

Laborationsuppgifter–skall lämnas in

Uppgift 1

I en kommande matematikkurs får vi lära oss något om hur man approximerar en funktion $f(x)$ med speceiella polynom, s.k. Taylorpolynom.

Plotta funktionen $f(x) = \cos x$ tillsammans med de tre Taylorpolynomen

$$P_2(x) = 1 - \frac{x^2}{2}, \quad P_4(x) = 1 - \frac{x^2}{2} + \frac{x^4}{24}, \quad P_6(x) = 1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720}$$

för $-2\pi \leq x \leq 2\pi$.

Föreskrifter: Funktionen $f(x)$ ska vara heldragen i blå, medan $P_2(x)$ ska vara heldragen i rött, $P_4(x)$ ska vara heldragen i grönt och $P_6(x)$ ska vara heldragen i magenta, Gör plotten av de 4 kurvorna tydlig genom att använda lämplig förklaring som titel, 'legends' etc.

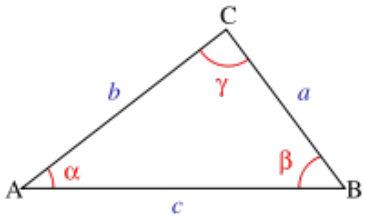
Samla funktioner/kommandon i en scriptfil med namnet `uppg1.py` och kör filen genom att anropa den från Ipython-konsolen i Spyder.

Uppgift 2

Den grekiske matematikern Heron upptäckte en algoritm för att kunna bestämma arean A av en triangel. Herons algoritm beskrivs av följande formel

$$A = \frac{1}{4}\sqrt{4a^2b^2 - (a^2 + b^2 - c^2)^2}$$

där a, b, c är triangelns tre sidor enligt nedanstående figur.



Skriv en funktion som bestämmer arean av en triangel med Herons's formel.
Använd funktionen för att bestämma arean av triangeln med sidorna

(a) $a = 5, b = 7, c = 9$

(b) $a = 3, b = 4, c = 5$.

Samla funktion/kommandon i en scriptfil med namnet `uppg2.py` och kör filen genom att anropa den från Ipython-konsolen i Spyder.

Uppgift 3

Bestäm med Python vinkeln mellan vektorerna

$$\mathbf{u} = \begin{bmatrix} 0 \\ 4 \\ -3 \end{bmatrix} \quad \text{och} \quad \mathbf{v} = \begin{bmatrix} -2 \\ 1 \\ 5 \end{bmatrix}$$

Observera: Svaret skall anges i *grader*.

Samla funktion/kommandon i en scriptfil med namnet `uppg3.py` och kör filen genom att anropa den från Ipython-konsolen i Spyder.