ma209A lab 1

Per Jönsson och Stefan Gustafsson
Fakulteten för Teknik och Samhälle, 2022

Viktig information om laborationerna

Laborationsdelen på kursen ma209A examineras genom att man gör två datorlaborationer. Obligatorisk närvaro gäller vid båda laborationstillfällena. För att bli godkänd på labbkursen krävs att båda laborationerna har redovisats på ett godtagbart sätt. Vid laborationerna gäller följande:

- Laborationsuppgifterna skall vara gjorda innan man kommer till laborationen (du är alltså tvungen att sitta hemma eller i datorsal före laborationen och göra uppgifterna samt förbereda dig). Har du stött på problem med uppgifterna kan du fråga din övningsledare, kursledaren eller någon kursare.

- Under laborationen skall uppgifterna redovisas. Under laborationen kan du även få hjälp med något moment du inte har lyckats få rätt på hemma. Studenter som inte har förberett uppgifterna när de kommer till laborationen underkänns och får göra om labbkursen i dess helhet nästa år.

- Samla lösningar och utskrifter samt figurer i ett dokument, (t.ex. ett word dokument), så att det går lätt att redovisa.

- Ni redovisar i grupper om tre med 10 min för varje redovisning. Inför redovisningen ska samtliga deltagare ha förberett ett dokument enligt ovan och kunna svara på frågor från labbhandledaren.

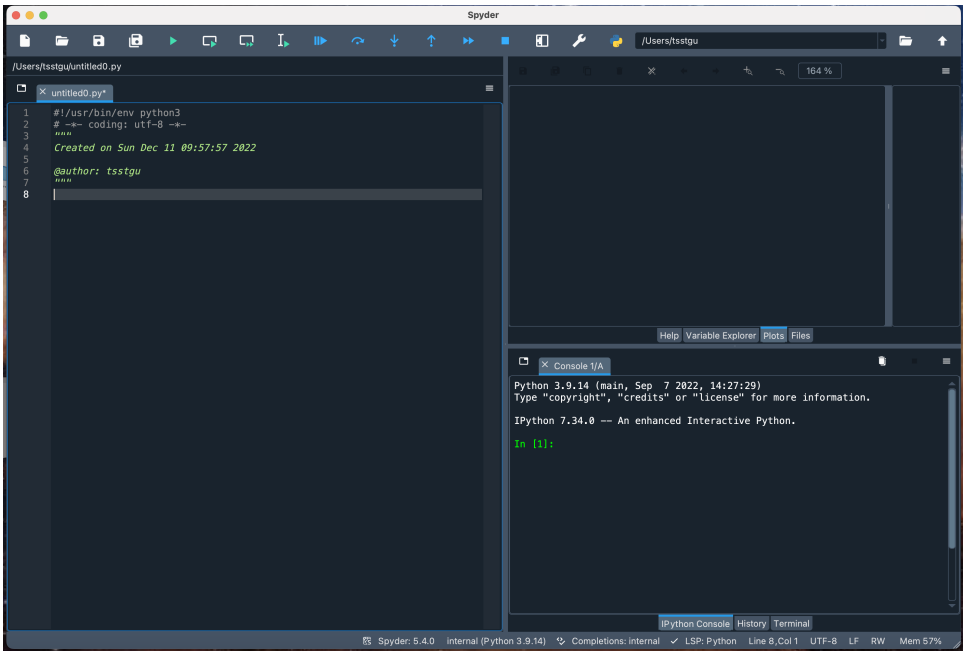
- Vid laborationstillfälle 1 redovisas laboration 1, vid laborationstillfälle 2 redovisas laboration 2 osv. Man kan t.ex. inte utebli under laboration 1 och sedan redovisa laboration 1 och 2 vid laborationstillfälle 2. Det går inte att byta laborationsgrupp under kursens gång.

- Om man inte kan närvara på en laboration på grund av sjukdom så måste detta anmälas snarast till kursledaren. För de som har anmält frånvaro på grund av sjukdom till kursledaren finns två reservtillfällen då man kan redovisa laborationer man missat. Endast studenter som har meddelat frånvaro till kursledaren på grund av sjukdom bereds plats vid reservtillfällena.

Reglerna ovan tolkas strikt och är till för att få laborationsmomentet att fungera praktiskt och underlätta er egen planering

1 Python som miniräknare

Spyder är ett användargränssnitt som är rekommenderat att ni använder under labbarna.



Figur 1: Spyders användargränssnitt

Med Spyder kan Python användas som en avancerad miniräknare.
De vanliga räkneoperationerna är definierade enligt tabellen nedan

**	potens (upphöjt till)	prioritet 1
*	multiplikation	prioritet 2
/	division	prioritet 2
+	addition	prioritet 3
-	subtraktion	prioritet 3

Då två räkneoperationer har samma prioritetsordning utförs beräkningarna från vänster till höger. Observera att man måste använda decimalpunkt i stället för decimalkomma. Stora och små tal skrivs lättast i exponentform. Så skriver man

7.51 e-6 i stället för 7.51×10^{-6}
8.32 e11 i stället för 8.32×10^{11}.

Tal kan med fördel lagras i variabler som man använder då man räknar.

Exempel 1.
(a) För att addera talen 12 och 13 skriver vi i IPython Console (rutan nere till höger, se Figur 1)

```
In [1]: 12+13
```

Resultatet läggs i variabeln Out[1]: som skrivs ut på skärmen

```
Out[1]: 25
```

(b) För att beräkna $2 \cdot 3^2$ skriver vi

```
In [2]: 2*3**2
```

Eftersom ** har högst prioritet börjar Python med att beräkna 3^2. Sedan följer multiplikation med 2 och Python svarar

```
Out[2]: 18
```

(c) Division och multiplikation har samma prioritet. Då vi skriver

```
In [3]: 1/2*3
```

börjar Python från vänster och beräknar $1/2$ sedan följer multiplikation med 3, vilket ger svaret

```
Out[3]: 1.5
```

För att undvika missförstånd bör man sätta ut parenteser och istället skriva talet som (1/2)*3.
(d) Då vi skriver

```
In [4]: 3.18**9
```

returnerar Python svaret

```
Out[4]: 33254.014350050005
```

(e) För att lagra talet 1.55×10^{-14} i variabeln x ger vi kommandot

```
In [5]: x = 1.55e-14
```

Skriver vi

```
In [6]: x
```

returnerar Python svaret

```
Out[6]: 1.55e-14
```

(f) Den kinetiska energin hos en kropp med massan m och farten v ges av uttrycket

$$Wkin = \frac{1}{2}mv^2$$

Vi har en bil med massan 1200 kg och farten 20 m/s. Vi inför variablerna m och v och beräknar energin enligt nedan

In [7]: m = 1200

In [8]: v = 20

In [8]: W = (1/2)*m*v**2

Skriver vi

In [9]: W

svarar Python

Out[9]: 240000.0

2 Matematiska funktioner

Standard Python saknar många av de grundläggande funktionerna som vi använder i matematiken. Dessutom saknar standard Python stöd för att hantera problem ur den linjära algebran. Modulen NumPy ger stöd för att hantera vektorer och matriser och innehåller dessutom de vanliga standard funktionerna. Där finns en mängd kommandon för matematiska funktioner i biblioteket Numpy av vilka en del är listade nedan.

np.absolute(x)	ger absolutbeloppet |x|.
np.sqrt(x)	ger kvadratroten $\sqrt{x}$.
np.exp(x)	ger exponentialfunktionen e^x.
np. log(x)	ger naturliga logaritmen $\ln x$.
np.log10(x)	ger 10-logaritmen log_10x.
np.sin(x)	ger $\sin x$ där x är i radianer.
np.cos(x)	ger $\cos x$ där x är i radianer.
np.tan(x)	ger $\tan x$ där x är i radianer.
np.cot(x)	ger $\cot x$ där x är i radianer.
np.arcsin(x)	ger $\arcsin x$ i radianer.
np.arccos(x)	ger $\arccos x$ i radianer.
np.arctan(x)	ger $\arctan x$ i radianer.
np.deg2rad(x)	omvandlar vinkeln x från grader till radianer.

I de trigonometriska funktionerna ska vinkeln ges i radianer. Vill man ange vinkeln i grader istället får man konvertera från grader till radianer med kommandot i tabellen ovan.
Numpy måste importeras innan funktionerna som finns i detta bibliotek kan användas. Vanligtvis importeras Numpy med namnet np enligt

In [1]: import numpy as np

Exempel 2.
(a) Talet π betecknas i NumPy med np.pi. Om vi till exempel skriver

In [2]: np.cos(np.pi)

får vi

Out[2]: -1.0

(b) För att få cosinusvärdet för 60 grader ger vi kommandot

In [3]: np.cos(np.deg2rad(60))

och får svaret
Out[3]: 0.5000000000000001

(c) Uttrycket $\sqrt{2}$ beräknas genom

In [4]: np.sqrt(2)

och Python returnerar

Out[4]: 1.4142135623730951

3 Vektorer

En vektor **x** är en samling av reella eller komplexa tal **x**= (x1,x2,...,xn). Längden av vektorn är lika med antalet element. Observera att det datalogiska begreppet vektorlängd inte skall blandas samman med det geometriska begreppet vektorlängd, vilket är definierat som kvadratroten ur summan av elementens kvadrater. En vektor som har längden ett kan identifieras med ett vanligt tal.

I Python med modulen NumPy importerad fås vektorer genom att skriva elementen inom hakparentes

x = np.array([x1,x2, ... ,xn])

Exempel 3.
För att generera vektorn x= (5 ,-4,6) kan man skriva

x = np.array([5,-4,6])

Skriver man print(x) får man svaret
[5 -4 6]

Vektorer där elementens värden ligger på konstanta avstånd är mycket vanliga inom programmering, och Python har ett antal kommandon för att generera sådana vektorer.

np.arange(a,b)	ger en vektor med värden från a till det största talet som är mindre än b i steg om ett. Om a> b fås en tom vektor.
np.arange(a,b,h)	ger en vektor med värden från a till det största talet som är mindre än b i steg om h, där h kallas steglängden. Det är tillåtet med negativ steglängd, i detta fall ger kommandot en vektor med värden från a till det minsta talet som är större än b.
np.linspace(a,b)	ger en vektor med hundra element jämnt fördelade mellan a och b.
np.linspace(a,b,n)	ger en vektor med n element jämnt fördelade mellan a och b.

I nedanstående exempel skrivs inte In [n]: eller Out[n] ut.

Exempel 4.

(a) Då vi skriver

x = np.arange(20,24)

genereras en vektor med talen 20 21 22 23.

Skriver vi print(x)
svarar Python [20 21 22 23]
]
Steglängden är ett och behöver inte sättas ut.

(b) Då vi matar in x = np.arange(20,22.2,0.5)
genererar Python en vektor med talen 20.0 20.5 21.0 21.5 22.0.

Skriver vi print(x)

svarar python [20. 20.5 21. 21.5 22.]

Steglängden är 0.5.
Observera att det sist genererade elementet är mindre än högergränsen 22.2.

(c) Även negativa steglängder är tillåtna. Till exempel ger x=np.arange(10,0,-2)
en vektor med talen 10 8 6 4 2
Med
print(x)

svarar python

[10 8 6 4 2]

(d) Kommandot

x = np.linspace(0,1,11)

ger en vektor x med 11 element jämnt fördelade mellan 0 och 1. Vektorn är densamma som

den man får genom

x = np.array([0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0])

4 Plottning

För att plotta en matematisk funktion $y = f(x)$ i intervallet $[a, b]$ börjar man med att generera ett antal x-värden jämnt fördelade mellan a och b och lagra dem i en vektor $\mathbf{x}$.
För varje x-värde räknar man sedan ut motsvarande y-värde och lagrar dem i en vektor $\mathbf{y}$.
Det grafiska biblioteket Matplotlib innehåller en hel del rutiner för att skapa en plot.
Vi tar några exempel och kommer här att skriva in kommandona i en fil istället.

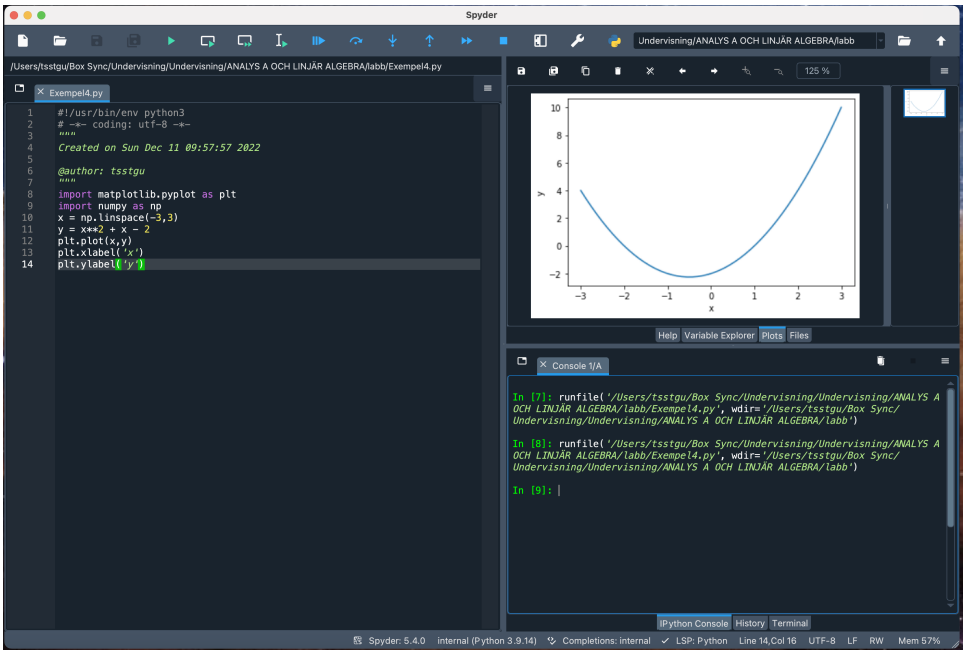
Exempel 5.

(a) För att plotta funktionen $y = x^2 + x - 2$ i intervallet $[-3, 3]$ ger vi kommandona

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-3,3)
y = x**2 + x - 2
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('y')
```

Kommandona plt.xlabel och plt.ylabel används för att sätta ut text på axlarna.

Filer skrivs in i Spyders användargränssnitt i rutan till vänster, se Figur 2.
Spara filen med tredje ikonen och kör med femte ikonen i den grå menyraden ovanför den vänstra rutan. Den genererade plotten visas i övre högra rutan.

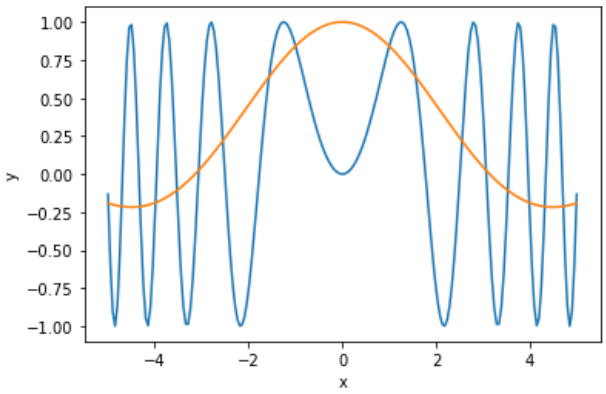


Figur 2: *Plottning med Spyder exempel 5(a)*

(b) För att plotta de två funktionerna $y = \sin(x^2)$ och $y = \frac{\sin(x)}{x}$
med x i intervallet [-5,5] ger vi kommandona i en fil och kör denna

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-5,5)
y1 = sin(x**2)
y2 = sin(x)/x
plt.plot(x,y1)
plt.plot(x,y2)
plt.xlabel('x')
plt.ylabel('y')
```

Den genererade plotten visas i Figur 3



Figur 3: *Plottning med Python exempel 5(b)*

5 Matematisk analys - att arbeta med funktioner

Vi ska nu övergå till matematisk analys och börjar diskussionen med hur man kan definiera egna funktioner i Python. Denna typ av funktioner kallas för anonyma funktioner, (eller lambda funktioner i Python).

funtionsnamn = lambda x,y,...: funktionsuttryck

Exempel 6. Vi inför en funktion $f(x) = x^2$ genom att ge kommandot

f = lambda x:x**2

Då vi väl infört funktionen kan vi anropa den på vanligt sätt.

Då vi till exempel skriver f(2) svarar Python 4

Vi kan anropa funktionen med en vektor. Om vi skriver

```
x = np.array( [-2,-1,0,1,2] )
y = f(x)
print(y)
```

får vi resultatet [4 1 0 1 4]

dvs funktionsvärdet har beräknats för vart och ett av värdena i vektorn **x**.

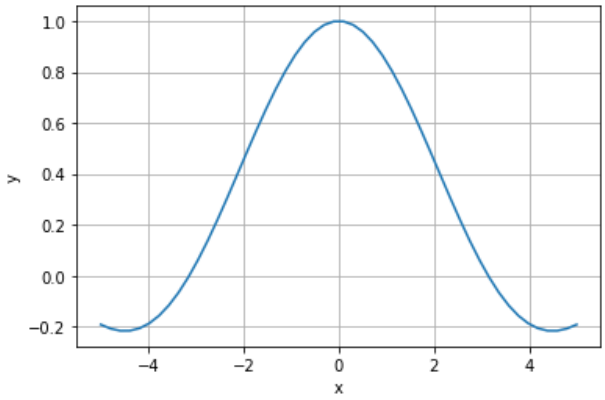
Exempel 7. Vi inför en funktion $g(x) = \dfrac{\sin(x)}{x}$
genom att ge kommandona

```
import numpy as np
g = lambda x:np.sin(x)/x
```

För att plotta funktionen i intervallet $[-5, 5]$ genererar vi en vektor **x** med värden jämnt fördelade mellan -5 och 5. Vi beräknar sedan funktionsvärdena y= g(x) för vart och ett av värdena i vektorn x och anropar sedan plotkommandot

```
x = np.linspace(-5,5)
y = g(x)
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
```

Vi får då plotten i figur 4. Notera att kommandot grid on ger oss hjälplinjer i plotten.



Figur 4: *Plottad funktion i Python. Man kan enkelt klippa ut en plott och klistra in den i ett Worddokument eller så kan man spara den som jpg-bilder.*

6 Nollställen, maximipunkter och integraler

I Python finns det kommandon med vars hjälp man på ett mycket enkelt sätt kan bestämma nollställen och minimipunkter. Nedanstående rutiner finns i biblioteket scipy.optimize.newline, som vi
som vi importerar genom
`import scipy.optimize as optimize`

Dessa kommandon sammanfattas nedan.

optimize.fsolve(f,x0)	bestämmer nollstället till funktionen f. x0 är. ett startvärde som skall ligga nära det sökta nollstället
optimize.minimize(f,x0)	ger x-värdet för ett lokalt minimum till funktionen f närheten av startvärdet x0.

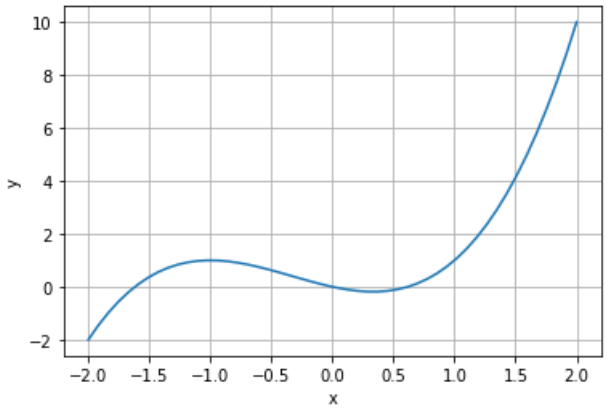
Exempel 8. Vi har en funktion
$y = f(x) = x^3 + x^2 - x$ $x \in [-2, 2]$. Funktionen definieras genom

```
import numpy as np
f= lambda x:x**3+x**2-x
```

För att plotta funktionen i intervallet [-2,2] ger vi kommandona

```
x = np.linspace(-2,2)
y = f(x)
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
```

och vi får plotten i Figur 5.
Från figuren ser vi att vi har ett nollställe i närheten av $x = -1.5$



Figur 5: *Funktionen* $y = f(x) = x^3 + x^2 - x$.

och en lokal minpunkt nära $x = 0.5$ samt en lokal maxpunkt nära $x = -1.0$.
För att bestämma nollstället närmare ger vi kommandona (observera att vi matar in tal med decimalpunkt och inte decimalkomma!)

```
import scipy.optimize as optimize
x0= optimize.fsolve(f,-1.5)
print(x0) ger utskriften [-1.61803399]
```

Minpunkten fås genom (obs decimalpunkt)

```
xmin = optimize.minimize(f,0.5)
```

print(xmin.x)
och vi får utskriften [0.33333333]

Obs! Med xmin.x får man endast minimipunkten utskriven.

Om man vill bestämma en maxpunkt för en funktion använder man igen kommandot optimize.minimize, men anger funktionen med negativt tecken.
För att bestämma maxpunkten för funktionen ovan i närheten av $x = -1.0$
ger vi kommandona
import numpy as np
fn = lambda x:-(x**3+x**2-x)
xmax = optimize.minimize(fn,-1.0)
print(xmax.x)
Utskriften blir [-0.9999998]

För numerisk integration av funktioner importerar vi biblioteket scipy.integrate.
I detta bibliotek finns integrationsrutinen quad(f,a, b), där f är integranden och a och b är nedre respektive övre gräns.

Exempel 9. Integralen $\int_0^2 x\cos(x)dx$ beräknas med följande kommandon

import scipy.integrate as integrate
f = lambda x:x*np.cos(x)
I=integrate.quad(f, 0, 2)
print(I)

Utskriften blir (0.40244801710422096, 8.177956603010255e-15)
Den första siffran är det beräknade värdet av integralen och den andra är noggrannheten med vilken integralen beräknats.

7 Uppgifter att redovisa

Nedanstående uppgifter skall redovisas under laborationstillfället. Observera att uppgifterna skall göras hemma innan laborationen och att det är redovisning som gäller under laborationen. Om det har funnits uppgifter som du har kört fast på så kan du få hjälp med dessa vid laborationstillfället. Python kommandona som behövs för att lösa uppgifterna kopierar du till ett Worddokument eller liknande så att det går att följa vad du gjort. Klipp även in Pythons svarsutskrifter och eventuella plottar eller figurer. Du skall visa upp dokumentet med kommandon och svarsutskrifter för din laborationshandledare i samband med redovisningen. Du skall också vara beredd på att svara på frågor kring hur du har löst uppgifterna. Se till att svara på alla uppgifterna.

1. Använd Python för att beräkna
   (a) $35 \cdot 25$ (b) $\sqrt{5} + 1/4$ (c) e^{10} (d) $\tan(\pi/6)$.

2. Du har följande utskrift från Python (a) 1.3533e+16 (b) 2.1191e-11.
   Skriv båda talen i vanlig form med tiopotenser.

3. Plotta funktionerna $y = \sin(x)$ och $y = \sin(x)\cos(x)$ i intervallet $[0, 10]$.
   Bägge funktionerna skall plottas i samma figur.

4. Använd kommando av typen np.arange(a,b,h) för att generera en vektor med värdena

   (a) 10 12 14 16 18 20 22 24 26 28 30 32
   (b) -2 -5 -8 -11 -14 -17 -20 -23 -26 -29 -32 -35

   Glöm inte att importera numpy och matplotlib.pyplot

5. Inför funktionen $f(x) = 1.2x + \sqrt{x} - 2,\ 0 \leq x \leq 2$.
   Plotta funktionen i intervallet $[0, 2]$, sätt ut gridlinjer och skriv axeltext.
   Använd kommandot optimize.fsolve för att bestämma funktionens nollställe.
   Observera att du måste importera biblioteket scipy.optimize
   för att kommandot skall fungera, se Exempel 8.
   När du matar in ett startvärde använd decimalpunkt!

6. Inför funktionen $f(x) = e^{-x}sin(x),\ 0 \leq x \leq 6$. Plotta funktionen i intervallet $[0, 6]$,
   sätt ut gridlinjer och skriv axeltext. Använd kommandot optimize.minimize för att
   bestämma funktionens minpunkt när x= 4.
   Observera att du måste importera biblioteket scipy.optimize
   för att kommandot skall fungera, se Exempel 8.

7. Beräkna integralen $\int_0^5 (x^3 + 2x^2 - x + 1)dx$
   med hjälp av kommandot quad.integrate.
   Kontrollera värde genom att beräkna integralen för hand
   (obs denna räkning skall också redovisas under labben).
   Observera att du måste importera biblioteket scipy.integrate,
   se Exempel 9.