

Python i MEK1100

En oversettelse fra Matlab til Python av deler av kompendiet

Feltteori og vektoranalyse

av

Bjørn Gjevik og Morten Wang Fagerland
2016

oversettelse ved
Karsten Trulsen

med bistand fra
Susanne Støle-Hentschel og Hans Petter Langtangen

Seksjon for Mekanikk
Matematisk institutt
Universitetet i Oslo
2016

Kapittel 3

Bruk av Python

3.1 Innledning

I det følgende har vi oversatt kapittel 3 i kompendiet Gjevik & Fagerland (2016) fra Matlab til Python. Mens kompendiets kapittel 3 er skrevet uten å anta at man kan noe om Matlab fra før, vet vi at dagens typiske student i MEK1100 har forkunnskaper i Python som går langt forbi den introduksjonen som gis her. Vi har likevel prøvd å oversette kapittel 3 i kompendiet forholdsvis “ordrett” for at man lettere kan få et inntrykk av likheter og forskjeller i de to språkene.

For grunnleggende introduksjon til Python anbefaler vi lærebøkene i INF1100 (Langtangen, 2014) samt Python-versjonen (Ryan, 2015) av MATLAB-appendikset i læreboka i MAT1110 (Lindstrøm & Hveberg, 2015).

Vi skal her basere oss på Matplotlib for å generere grafikk fordi dette har vært etterspurt av studenter i MEK1100 tidligere semester. For å lære mer om Matplotlib kan vi anbefale nettsiden <http://matplotlib.org/>.

3.2 Litt grunnleggende Python

3.2.1 Oppstart, grensesnitt, kommentarer og pakker/moduler

På Linux og Unix kan Python startes ved å skrive `python` eller `ipython` i et terminalvindu.

I terminalvinduet kan du skrive inn Python-kommandoer som vil bli eksekvert med en gang du trykker på Enter. I `python` betyr symbolet `>>>` at Python er klar til å ta imot en kommando. I `ipython` betyr symbolet `In [#]:` (hvor `#` er et tall) det samme. Resultatet av denne vil bli skrevet ut rett under (se eksempler i neste avsnitt).

Når du skal skrive Python-kode er det veldig lurt (som i alle andre programmeringsspråk) å venne seg til å skrive kommentarer slik at det er lettere både for deg selv og for andre å lese koden ved en senere anledning. Linjekommentarer i Python starter med et `#` og alt fra dette tegnet og ut linjen vil oppfattes som kommentar. Kommentarer over flere linjer er definert ved `'''` (3 tegn) før og etter kommentaren.

Importere numpy for numerikk

Matematiske konstanter og funksjoner som kan ta argumenter som er både skalarer, lister og arrayer gjøres tilgjengelig fra `numpy`:

```
>>> import numpy as np
>>> print np.pi, np.sin(1)
3.14159265359 0.84147098
>>> np.sin([1, 2]), np.sin(np.array([1, 2]))
[ 0.84147098  0.90929743] [ 0.84147098  0.90929743]
```

Her merker vi oss at “[1, 2]” er ei liste mens “`np.array([1, 2])`” er et objekt av type `numpy.ndarray`, sistnevnte er å foretrekke for numerisk arbeid. `array()` diskuteres i seksjon 3.2.2.

Importere matplotlib.pyplot for grafikk

Vi skal bruke `matplotlib.pyplot` for å generere grafikk. Følgende tre linjer tegner opp ei rett linje mellom punktene (0,1) og (1,2):

```
import matplotlib.pyplot as plt
plt.plot([0, 1], [1, 2])
plt.show()
```

Avhengig av hvordan Python har blitt startet opp, dersom Python ikke er i interaktiv modus så vil kommandoen `show()` være nødvendig for at grafikkvinduet skal komme opp på skjermen.

Interaktiv modus

For den typen arbeid vi skal gjøre i dette kurset vil det være behagelig at grafikken på skjermen blir oppdatert etterhver som vi gir kommandoer. Dette kan oppnås ved å gi kommandoen `interactive(True)`. I følgende eksempel vil således kommandoen `show()` ikke være nødvendig:

```
import matplotlib.pyplot as plt
plt.interactive(True)
plt.plot([0, 1], [1, 2])
```

Interaktiv modus kan også slås på og av med kommandoene `ion()` og `ioff()`, og man kan sjekke modus med kommandoen `isinteractive()`.

Importere pylab for både numerikk og grafikk

For praktisk arbeid i dette kurset vil det være hensiktsmessig å importere `pylab` som medfører at både `numpy` og `matplotlib.pyplot` blir importert. Dessuten er det enklere å ikke måtte skrive forstavelsene “`np.`” og “`plt.`” hele tiden. Dette kan vi oppnå ved:

```
from pylab import *
```

Utskrift av grafikk til fil

Den enkleste måten å plote ut grafikk til fil med Matplotlib er å bruke det grafiske grensesnittet i plottevinduet. Klikk på “Save the figure”, velg katalog og filnavn å skrive til samt filtype, klikk på “Save”.

Utskrift av grafikk til fil kan også gjøres med `savefig("filnavn.filtype")`. Fila vil bli skrevet i det formatet som angis av `filtype`. Her er et eksempel på å skrive et tomt koordinatsystem til fila `utskrift.pdf`:

```
from pylab import *
plot([0, 1], [1, 2])
savefig("utskrift.pdf")
show()
```

Det er verdt å merke at dersom Python kjøres i ikke-interaktiv modus må sannsynligvis `savefig` komme før `show` for å få lagret grafikken til fila. Dersom Python derimot kjøres i interaktiv modus så er sannsynligvis ikke `show` nødvendig.

3.2.2 Matriser

En matrise kan lages med kommandoen `array` som hører til modulen `numpy`. Objektet som returneres er av type `numpy.ndarray`, som er å foretrekke framfor vanlige lister for numerisk arbeid. Elementene separeres med *komma* (`,`). Hver rad skal rammes inn med *klammeparenteser* (`[]`). Hele matrisen skal også rammes inn med *klammeparenteser* (`[]`). Resultatet kan vises med kommandoen `print`.

```
>>> from pylab import *
>>> A = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> print A
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Det er flere måter å transponere matrisen, for eksempel ved å bruke kommandoen `transpose`.

```
>>> print A.transpose()
[[1 4 7]
 [2 5 8]
 [3 6 9]]
>>> print transpose(A)
[[1 4 7]
 [2 5 8]
 [3 6 9]]
>>> print A.T
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

For å finne den inverse til en matrise har vi funksjonen `linalg.inv`.

```
>>> B = array([[1,3,0],[2,0,4],[1,3,1]])
>>> print B
[[1 3 0]
 [2 0 4]
 [1 3 1]]
>>> C = linalg.inv(B)
>>> print C
[[ 2.          0.5        -2.          ]
 [-0.33333333 -0.16666667  0.66666667]
 [-1.          0.          1.          ]]
```

Vi sjekker deretter at produktet av matrisene B og C gir enhetsmatrisen

```
>>> print dot(C,B)
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

Funksjonen `diag` returnerer matrisens diagonal og `sum` summerer opp kolonnene.

```
>>> print diag(A)
[1 5 9]
```

Funksjonen `sum` summerer opp kolonnene eller radene avhengig av verdien til `axis`-parameteren (0=rad, 1=kolonne).

```
>>> print sum(A, axis=0)
[12 15 18]
>>> print sum(A, axis=1)
[ 6 15 24]
```

Elementet i rad i og kolonne j av A er $A[i, j]$ (husk at i Matlab starter indekseringen fra 1, mens i Python starter indekseringen fra 0).

```
>>> print A[1,0] + A[2,1]
12
```

```
>>> A[0,2] = 0
>>> print A
[[1 2 0]
 [4 5 6]
 [7 8 9]]
```

Kommandoen `shape` returnerer antall rader og kolonner til en matrise.

```
>>> print shape(A)
(3, 3)
```

Kommandoen `size` returnerer antall elementer i en matrise

```
>>> print size(A)
9
```

Det finnes flere Python-kommandoer som gjør det enkelt å generere matriser. Vi har sett at funksjonen `diag(A)` returnerer `A`s diagonal, men den kan også brukes til å bygge opp en matrise:

```
>>> A = diag([1,2,3])
>>> print A
[[1 0 0]
 [0 2 0]
 [0 0 3]]
```

Andre nyttige kommandoer som kan nevnes er `ones`, `zeros` og `eye` som lager matriser med henholdsvis enere, nuller, og identitetsmatrisen. Du kan lese mer om dem ved å bruke `help` (f.eks. `help(zeros)`).

Matematiske operasjoner på arrayer

Det matematiske funksjonene i `numpy` kan operere komponentvis på arrayer

```
>>> from numpy import *
>>> A = array([[1, 2]])
>>> print sin(A)
[[ 0.84147098  0.90929743]]
```

3.2.3 Kolon-operatoren, `arange` og `linspace`

For å generere en vektor med heltallige verdier fra en aritmetisk tallfølge kan vi bruke `arange`. For eksempel, en vektor (tallrekke) fra -5 til 5 med intervall 1

```
>>> from numpy import *
>>> a = arange(-5,6)
>>> print a
[-5 -4 -3 -2 -1  0  1  2  3  4  5]
```

Det genereres tall større eller lik første argument, og mindre enn andre argument. En vektor fra -15 til 15 med intervall 3

```
>>> a = arange(-15, 16, 3)
>>> print a
[-15 -12  -9  -6  -3   0   3   6   9  12  15]
```

For ikke-heltallige verdier anbefales det å bruke `linspace` for å unngå avrundingsfeil, `linspace` genererer så mange tall som tredje argument jevnt fordelt mellom første og andre argument

```
>>> a = linspace(-15, 15, 11)
>>> print a
[-15. -12.  -9.  -6.  -3.   0.   3.   6.   9.  12.  15.]
```

Det finnes en annen kommando `range` som fungerer tilsynelatende likt som `arange` for heltallige argumenter, men mens `arange` genererer objekter av type `numpy.ndarray` så genererer `range` ei liste som ikke anbefales for numerisk arbeid.

I Python kan kolon-operatoren brukes til å indeksere matriser og vektorer, og inkluderer alt men ikke det siste tallet

```
>>> A = array([[1, 2, 3, 4],[ 5, 6, 7, 8],[ 9, 10, 11, 12],[ 13, 14, 15, 16]])
>>> print A
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
>>> print A[0:3,1:4]
[[ 2  3  4]
 [ 6  7  8]
 [10 11 12]]
```

Rad 2 og 3, alle kolonnene

```
>>> print A[1:3,: ]
[[ 5  6  7  8]
 [ 9 10 11 12]]
```

Steglengde 2 langs kolonnene

```
>>> print A[1:3,1:4:2]
[[ 6  8]
 [10 12]]
```

Steglengde 3 langs kolonnene

```
>>> print A[1:3,0:4:3]
[[ 5  8]
 [ 9 12]]
```

Det burde nå være åpenbart at kolon-operatoren fungerer på dramatisk forskjellig måte i Matlab og Python!

3.2.4 py-filer og funksjoner

Isteden for å taste inn kommandoer i terminal-vinduet kan vi samle kommandoer i en fil ved hjelp av en teksteditor og så kjøre filen i Python. Her antar vi at filen slutter på “.py” og kalles derfor en *py-fil*.

Her er en fil med kommandoer som definerer en matrise *A* og skriver ut noen egenskaper til matrisen:

```
from numpy import *
A = array([[1,3,0],[2,1,1]])
print A
print A.size
```

```
print A.shape
print diag(A)
print sum(A)
print A.transpose()
```

Denne filen kan du kalle hva som helst, la oss si at du kaller den `MatriseEksempel.py`, og den kan da kjøres ved å skrive “python `MatriseEksempel.py`” i et terminalvindu.

Under følger et eksempel på hvordan vi kan definere en funksjon som ligger i en egen fil ved navn `tridiag.py`:

```
from numpy import *

def tridiag(n):
    '''Konstruerer en nxn matrise med -2 paa
    diagonalen og 1 paa de subdiagonalene'''

    A = -2*eye(n) + diag(ones(n-1),1) + diag(ones(n-1),-1)
    return A
```

Linjen som starter med `def` forteller Python at dette er en funksjon som heter “tridiag” og tar en innparameter (n). De to neste linjene er en kort beskrivelse av funksjonen. Denne hjelpeteksten er alltid lurt å ta med ettersom det er denne teksten som `help` bruker for å beskrive funksjonen. Deretter beregnes matrisen A som returneres.

Funksjonen kan nå importeres:

```
from tridiag import *
```

Brukerveiledning fås med:

```
>>> help(tridiag)
Help on function tridiag in module tridiag:
```

```
tridiag(n)
    Konstruerer en nxn matrise med -2 paa
    diagonalen og 1 paa de subdiagonalene
```

Funksjonen kalles opp slik:

```
>>> print tridiag(5)
[[-2.  1.  0.  0.  0.]
 [ 1. -2.  1.  0.  0.]
 [ 0.  1. -2.  1.  0.]
 [ 0.  0.  1. -2.  1.]
 [ 0.  0.  0.  1. -2.]]
```


3.2.5 Enkel plotting i 2D

Kommandoen `plot` kan brukes til enkel plotting i to dimensjoner. La oss først plote funksjonen $y = x^3$ der x er definert over et område fra -2π til 2π :

```
from pylab import *
interactive(True)
x = linspace(-2*pi, 2*pi, 41)
y = x**3
plot(x,y)
```

Her har vi valgt å diskretisere x -intervallet med 41 punkt som gir steg på $\pi/10$. Velges dette steget for stort blir grafen hakkete. Prøv f.eks. med et steg på $\pi/2$.

Vi kan også enkelt plote en kurve som er gitt ved en parametrisering. En sirkel med radius r kan uttrykkes ved $x(t) = r \cos t$, $y(t) = r \sin t$ der $0 \leq t < 2\pi$. La oss plote en sirkel med radius 3:

```
from pylab import *
r = 3
t = linspace(0, 2*pi, 101)
x = r*cos(t)
y = r*sin(t)
interactive(True)
plot(x,y)
axis("equal")
```

For at sirkelen skal se ut som en sirkel og ikke som en ellipse, så er det et viktig poeng at en enhet langs x -aksen er like lang som en enhet langs y -aksen. Dette oppnås med kommandoen `axis("equal")` — prøv å utføre kommandosekvensen ovenfor uten å kalle `axis("equal")` og se at figuren da ser ut som en ellipse!

3.2.6 Elementvise operasjoner

Noen operasjoner (f.eks. multiplikasjon, divisjon og potens) kan man ønske å bruke på matriser på to forskjellige matematiske måter. Når variablene disse operasjonene virker på er skalarer, vil den vanlige betydningen (multiplikasjon, divisjon og potens) gjelde slik at f.eks. `3*2` vil gi det forventede svaret 6. Hvis derimot variablene er enten vektorer eller matriser, kan vi ønske at operasjonene enten er vanlige matriseoperasjoner eller elementvise operasjoner. I Matlab betyr `*`, `/`, `**` og `^` matriseoperasjoner (de to siste er synonymer for potens), mens man ved å tilføye et `.` foran operatorene får elementvise operasjoner. I Python betyr `*`, `/` og `**` elementvise operasjoner (`**` betyr potens, mens derimot `^` betyr noe helt annet). I eksempelet over der vi ønsker å regne ut $y = x^3$, og vi allerede har definert en vektor med x -verdier, vil vi at hvert element i x skal opphøyes i tredje; vi ønsker ikke å bruke matrisemultiplikasjonen av x med seg selv to ganger. Noen eksempler:

```
>>> a = arange(1,5)
>>> print a
[1 2 3 4]
```

```
>>> print 3*a
[ 3  6  9 12]
>>> print a*a
[ 1  4  9 16]
```

Indreprodukt beregnes med `dot`

```
>>> print dot(a,a)
30
```

Ytre eller dyadisk produkt beregnes med `outer`

```
>>> print outer(a,a)
[[ 1  2  3  4]
 [ 2  4  6  8]
 [ 3  6  9 12]
 [ 4  8 12 16]]
```

3.3 Plotting av flater

Vi skal nå se på hvordan vi kan lage plottet av flaten i figur 3.1. Dette er den samme flaten som vi har studert i figur 1.7 på side 19 i kapittel 1 i kompendiet og er gitt matematisk ved formelen:

$$h = \frac{h_0}{1 + \frac{x^2+y^2}{R^2}}. \quad (3.1)$$

Formelen modellerer en isolert fjelltopp der h er høyden over havflaten, h_0 er høyden på toppen av fjellet og R er et mål for radius i fjellet. Konstantene h_0 og R er her satt til 2277 m og 4000 m.

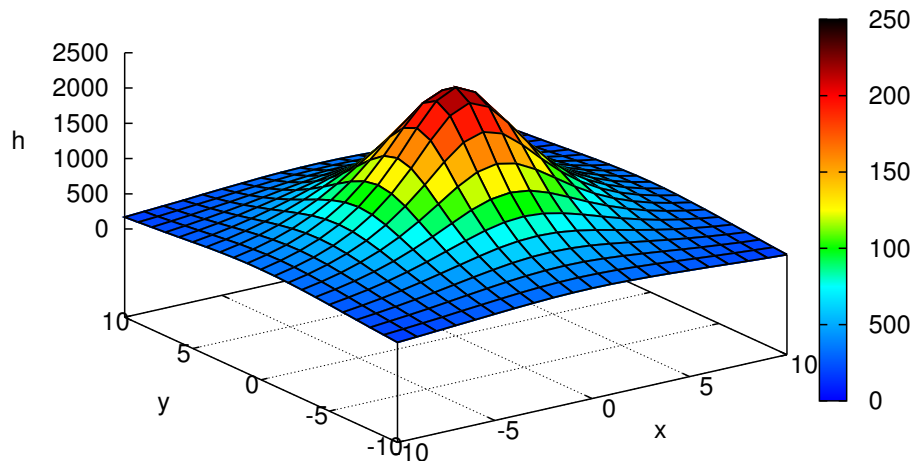
Det anbefales at du først prøver ut de påfølgende kommandoene direkte i Python for senere å samle dem i en py-fil.

Det første vi skal gjøre er å definere de to konstantene h_0 og R . Vi ser av figuren at x - og y -aksen er oppgitt i km og h -aksen i meter. Vi definerer derfor h_0 i meter og R i km:

```
h0 = 2277.
R = 4.
```

x - og y -aksen skal spenne over et område på 20×20 km med sentrum i origo. For å lage et passende grid med x - og y -verdier kan vi bruke kommandoen `meshgrid` som tar to vektorer som parametere. Vektorene vi bruker som innparametere kan vi lage ved å bruke `linspace`:

```
t = linspace(-10,10,41)
x,y = meshgrid(t,t, indexing="ij")
```



Figur 3.1: En modell av fjelltoppen Beerenberg på Jan Mayen.

Størrelsen på gridet må tilpasses hvert eksempel. Her har vi brukt 41 punkter per akse, noe som gir en passende maskestørrelse samtidig som flaten ser glatt ut.

Merk at `meshgrid` må angis med opsjon `indexing="ij"` for at etterfølgende operasjoner skal fungere korrekt: Prøv å lage figur 3.3 uten bruk av `indexing="ij"`.¹

Vi er nå klare til å regne ut h som angitt i (3.1):

$$h = h_0 / (1 + (x^2 + y^2) / (R^2))$$

For å tegne opp 3D grafikk skal vi bruke det såkalte Matplotlib `mplot3d` toolkit.² For å komme i gang kan vi først gjøre følgende:

```
from mpl_toolkits.mplot3d import axes3d
fig = figure()
ax = fig.add_subplot(111, projection="3d")
```

Deretter kan vi be om å få tegnet opp omtrent som Matlab `surf`

```
surf = ax.plot_surface(x, y, h, rstride=1, cstride=1, cmap=cm.jet,
                      linewidth=0, antialiased=False)
fig.colorbar(surf, shrink=0.5, aspect=5)
```

eller omtrent som Matlab `mesh` med skyggevirksomhet

¹2D grid kan organiseres på to måter: Kartesisk indeksering (standard `meshgrid` i Python/`numpy` og Matlab), eller matrise-indeksering (`meshgrid` med `indexing="ij"` i Python/`numpy`, `ndgrid` i Matlab og i Python/`scitools`). Merk at kommandoen `gradient` (se kapittel 3.5) ønsker kartesisk indeksering i Matlab og matrise-indeksering i Python.

²http://matplotlib.org/mpl_toolkits/mplot3d/

```
surf = ax.plot_surface(x, y, h, rstride=1, cstride=1, color="w",  
                      linewidth=0.1)
```

eller vi kan lage kurver som ikke har skyggevirking

```
ax.plot_wireframe(x, y, h, rstride=10, cstride=10)
```

For å endre fargene som skriv `cmap=cm.MAP` hvor MAP er et gyldig navn på colormap, for eksempel `jet`, `hot`, `bone`, `cool`, etc., dette står forklart i seksjon 3.4 om konturlinjer.

For å se hvilke farger som representerer hvilke verdier kan man skrive `colorbar()`, men det er selvfølgelig kun meningsfullt ved bruk av tilsvarende colormap.

Til slutt kan vi pynte litt på figuren. For å angi navn på aksene bruker vi

```
ax.set_xlabel("x")  
ax.set_ylabel("y")  
ax.set_zlabel("h")
```

Samler vi alle kommandoene i en py-fil blir den for eksempel seende slik ut:

```
from pylab import *  
from mpl_toolkits.mplot3d import axes3d  
interactive(True)  
h0 = 2277. # Hoyden av toppen av fjellet (m)  
R = 4.    # Maal for radius av fjellet (km)  
t = linspace(-10,10,21)  
x,y = meshgrid(t,t, indexing="ij")           # Grid for x- og y-verdiene (km)  
h = h0/(1 + (x**2+y**2)/(R**2))           # Beregn hoyden h (m)  
fig = figure()  
ax = fig.add_subplot(111, projection="3d")  
surf = ax.plot_surface(x, y, h, rstride=1, cstride=1, cmap=cm.jet,  
                      linewidth=0, antialiased=False)  
fig.colorbar(surf, shrink=0.5, aspect=5)  
ax.set_xlabel("x")  
ax.set_ylabel("y")  
ax.set_zlabel("h")
```

Her kan man bytte ut de to linjene med `plot_surface` og `colorbar` med alternativene vist ovenfor.

I eksemplet ovenfor er `show()` ikke nødvendig fordi vi kjører i interaktiv modus.³

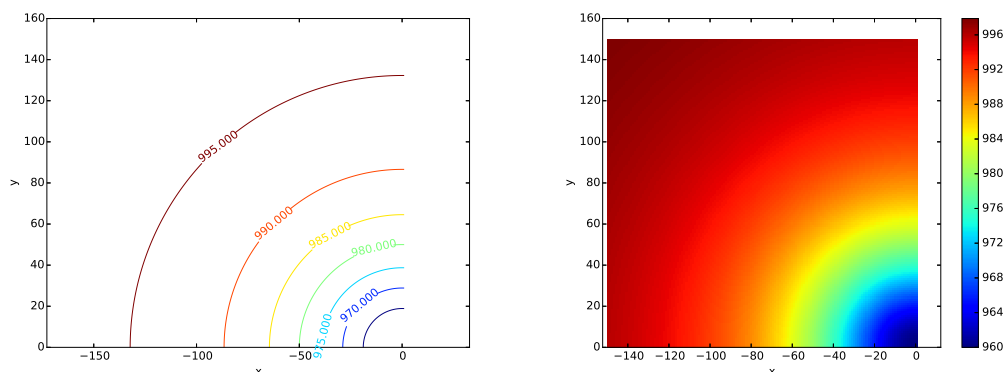
3.4 Plotting av konturlinjer

Vi skal ta for oss et nytt eksempel fra side 19 i kapittel 1 i kompendiet; lufttrykket ved havflaten i tilknytning til et stormsenter. Formelen vi skal bruke er gitt ved

$$p = p_0 - \frac{\Delta p}{1 + \frac{x^2+y^2}{R^2}} \quad (3.2)$$

³Dersom grafikk-kommandoer ikke ser ut til å gjøre sin effekt umiddelbart i interaktiv modus så må man kanskje “ta litt” på grafikk-vinduet med musa for å friske det opp. Det kan være nødvendig å “ta litt kraftig” på grafikk-vinduet, for eksempel ved å klikke på knappen “Configure subplots”.

der p_0 er lufttrykket langt borte fra sentrum, Δp er trykkfallet inn mot sentrum og R er et mål for utstrekningen av lavtrykket. Konturlinjene vi skal plote er gjengitt i figur 3.2.



Figur 3.2: Til venstre konturlinjer for konstant trykk (isobarer) rundt et lavtrykks-sentrum. Til høyre de samme med pcolor. Enhetene på aksene er kilometer.

Vi starter med konstantene som skal oppgis i km (R) og hPa (p_0 og Δp):

```
R = 50
p0 = 1000
dp = 40
```

Til forskjell fra kompendiet skal vi her la x -aksen være definert over et område fra -150 til 0 km, mens vi lar y -aksen være definert over et område fra 0 til 150 km, deretter regnes p -verdiene ut:

```
tx = linspace(-150, 0, 151)
ty = linspace(0, 150, 151)
x,y = meshgrid(tx,ty, indexing="ij")
p = p0 - dp/(1+(x**2+y**2)/R**2)
```

Konturlinjene kan nå plottes med kommandoen `contour`:

```
contour(x,y,p)
colorbar()
```

For å plote et valgfritt antall konturlinjer bruk `contour(x,y,p,n)` der n er antall linjer. Hvis vi kun er interessert i noen utvalgte trykkverdier (f.eks. 960, 970, 980 og 990) så kan vi gi verdiene som en vektor til `contour`:

```
v = [960, 970, 980, 990]
contour(x,y,p,v)
```

eller enda enklere: `contour(x,y,p,[960, 970, 980, 990])`.

Vil du bruke en annen fargeskala prøv `cmap=cm.MAP` hvor `MAP` er et gyldig navn for en "colormap", for eksempel `autumn`, `bone`, `cool`, `copper`, `flag`, `gray`, `hot`, `hsv`, `jet`, `pink`, `prism`, `spring`, `summer` eller `winter`.

For å se hvilke verdier linjene har kan man skrive eksplisitt på hver linje, og linjene bør da ha en mørk farge. Man kan benytte kommandoen `clabel` som vist nedenfor til å skrive på verdiene i plottet (legg merke til variabelen `C` som brukes til å overføre informasjon fra `contour` til `clabel`).

En py-fil for å generere figur 3.2 kan nå se slik ut:

```
from pylab import *
interactive(True)
R = 50      # Utstrekningen av lavtrykket (km)
p0 = 1000  # Lufttrykket langt borte fra sentrum (hPa)
dp = 40    # Trykkfallet inn mot sentrum (hPa)

tx = linspace(-150,1,151)
ty = linspace(0,150,151)
x,y = meshgrid(tx,ty, indexing="ij") # Grid for x- og y-verdiene (km)
p = p0 - dp/(1+(x**2+y**2)/R**2)     # Beregn trykket p (hPa)
C=contour(x,y,p)
clabel(C)
axis("equal")
xlabel("x") # Sett aksenavn
ylabel("y")
```

For et alternativt type plott som inneholder samme informasjon, prøv å bytte ut linjene med `contour` og `clabel` med `pcolor` og `colorbar`:

```
pcolor(x,y,p)
colorbar()
```

3.5 Plotting av vektorfelt og beregning av gradientvektor.

Et vektorfelt i kartesiske koordinater $\mathbf{v} = v_x(x, y)\mathbf{i} + v_y(x, y)\mathbf{j}$ kan plottes med

```
quiver(x,y,vx,vy, [faktor] , [farge])
```

der `faktor` og `farge` er valgfrie opsjoner som endrer henholdsvis vektorenes lengde med en skaleringsfaktor og vektorenes farge. La oss ta et eksempel:

$$\mathbf{v} = (x^2 + 2y - \frac{1}{2}xy)\mathbf{i} - 3y\mathbf{j}.$$

Vi lar x og y variere mellom -5 og 5 . Når vi skal definere et grid for x og y , må vi passe på å bruke et større intervallsteg enn vi vil bruke for overflateplott eller konturlinjer. Det blir fort veldig uryddig med for mange vektorer i en og samme figur.

```

from pylab import *
interactive(True)
t = linspace(-5,5,11)
x,y = meshgrid(t,t, indexing="ij")
vx = x**2 + 2*y - .5*x*y
vy = -3*y
quiver(x,y,vx,vy,scale=25,units="x",color="b") # skalareringsfaktor, blaa farge
axis("equal")

```

Her har vi skalert lengden til alle vektorene med en faktor 25 relativt til x -aksen som referanse, og vi har bedt om at vektorene skal tegnes i blå farge.⁴

Hvis vi har definert et skalarfelt $F(x,y)$ over et område utspent av vektorene x og y i Python kan vi regne ut gradientvektoren til feltet med `gradient(F)`. Denne funksjonen returnerer de partielt deriverte ($\partial F/\partial x$ og $\partial F/\partial y$ i 2D) til feltet. Med disse for hånden kan vi plote vektorfeltet med `quiver`:

```

dFx,dFy = gradient(F);
quiver(x,y,dFx,dFy)

```

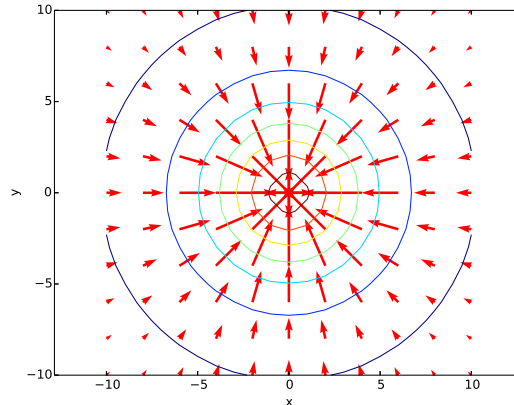
Vi skal videre presentere koden som er brukt for å lage figur 3.3 som er et plott av noen konturlinjer for h gitt ved (3.1) og gradientvektoren ∇h . Bortsett fra kommandoene `hold("on")` og `hold("off")`, som sørger for at konturlinjene og vektorfeltet blir plottet sammen og med samme akseegenskaper, skulle resten av koden nå være kjent.

```

from pylab import *
interactive(True)
h0 = 2277 # Hoyden av toppen av fjellet (m)
R = 4     # Maal for radius av fjellet (km)
tt = linspace(-10.,10.,11)
xx,yy = meshgrid(tt, tt, indexing="ij") # Definer et grovere grid til vektor
hh = h0/(1+(xx**2+yy**2)/(R**2)) # Beregn hoyden med det nye griddet
dhx,dhy = gradient(hh) # Beregn gradientvektoren (dh/dx,dh/dy)
# Plott vektorfeltet (rod farge) og skaler vektorlengden med en faktor
# En bedre skaleringsfaktor er .75, men fungerer kanskje ikke?
quiver(xx,yy,dhx,dhy)
hold("on") # Behold konturlinjene og akse-egenskapene
t = linspace(-10.,10.,21)
x,y = meshgrid(t,t, indexing="ij") # Grid for x- og y-verdiene (km)
h = h0/(1+(x**2+y**2)/(R**2)) # Beregn hoyden h (m)
contour(x,y,h) # Kontur og sett akseenhetene like
xlabel("x") # Sett aksnavn
ylabel("y")
axis("equal") # Sett akseenhetene like
hold("off") # Trenger ikke flere plott i denne figuren

```

⁴Det er forvirrende at i Matlab og Octave øker pilenes lengde med skaleringsfaktor, mens i Python/Matplotlib avtar pilenes lengde med skaleringsfaktor.



Figur 3.3: Konturlinjer for h og gradientvektoren ∇h . Enhetene på aksene er kilometer.

Merk: I denne figuren er det et viktig poeng at pilene skal stå vinkelrett på sirklene. Dette oppnår man ved å kreve at enhetene langs begge aksene er like, derfor er det viktig å ta med kommandoen `axis("equal")`.

Merk: Prøv å kjøre skriptet ovenfor uten å ta med de to forekomstene av `indexing="ij"` og legg merke til at figuren da blir helt feil!

3.6 Plotting av strømlinjer

I kapittel 4 møter vi strømfunksjonen ψ . Kjenner vi uttrykket for denne er det lett å plote strømlinjene. Siden strømfunksjonen er konstant langs en strømlinje vil strømlinjene være konturlinjer for skalarfeltet ψ . Vi kan derfor definere et grid i x - og y -retning med `meshgrid`, regne ut diskrete verdier av strømfunksjonen, og til slutt bruke `contour` for å tegne opp strømlinjene, akkurat på samme måte som vi plottet isobarene til trykkfeltet i seksjon 3.4.

Merk: Noen ganger ønsker vi å plote strømliner og ekviskalarflater i samme plott. I så fall er det et viktig poeng at de to kurvefamiliene står vinkelrett på hverandre. Dette oppnås ved å kreve at enhetene langs begge aksene er like, derfor er det viktig å ta med kommandoen `axis("equal")`.

3.7 Oppgaver

Oppgavene som står i kompendiet anbefales og kan gjøres uansett om man bruker Matlab eller Python.

Bibliografi

- GJEVIK, B. & FAGERLAND, M. W. 2016 *Feltteori og vektoranalyse*. Matematisk institutt, UiO.
- LANGTANGEN, H. P. 2014 *A Primer on Scientific Programming with Python*, 4th edn. Springer.
- LINDSTRØM, T. L. & HVEBERG, K. 2015 *Flervariabel analyse med lineær algebra*, 2nd edn. Gyldendal Akademisk Forlag.
- RYAN, Ø. 2015 Python-versjon av MATLAB-appendikset i “Flervariabel analyse med lineær algebra”.
<http://folk.uio.no/oyvindry/fvla/FVLAbokpythonappendix.pdf>.

Register

- Beerenberg, 10
- felt
 - trykkfelt, 15
- gradientvektor
 - figur med konturlinjer, 15
 - hvordan beregne med Python, 13–15
- isobarer, 12, 15
- konturlinjer, 15
 - figur, 12
 - figur med gradientvektor, 15
 - hvordan plote med Python, 11–13
- lavtrykk
 - i en lufttrykksmodell, 12
- lufttrykk
 - en modell som skal plottes, 11
- partiell derivasjon
 - hvordan beregne med Python, 14
- pcolor
 - figur, 12
- Python
 - , (komma), 3
 - : (kolon-operatoren), 6
 - [] (klammeparenteser), 3
 - #, 1
 - arange, 5, 6
 - axis, 8
 - beregning av gradientvektor, 13–15
 - beregning av skalarfelt, 9–11
 - clabel, 13
 - cmap, 11, 12
 - colorbar, 11–13
 - colormap, 12
 - contour, 12–13, 15
 - def, 7
 - diag, 4–5
 - dot, 9
 - elementvise operasjoner, 8–9
 - eye, 5
 - generering og manipulering av matriser, 3–5
 - gradient, 10
 - gradient, 13–15
 - grensesnitt, 1
 - grid, 10
 - help, 5, 7
 - hold on/off, 14
 - hvordan invertere en matrise, 4
 - hvordan transponere en matrise, 3–4
 - innledning, 1
 - interaktiv modus, 2
 - kolon-operatoren (:), 6
 - kommentarer, 1
 - konstanter/variable, 9, 12
 - linspace, 5
 - matematiske operasjoner på arrayer, 5
 - matematiske operasjoner på matriser, 5
 - Matplotlib, 1, 3
 - meshgrid, 9, 10, 12, 15
 - modulen `matplotlib.pyplot`, 2
 - modulen `numpy`, 2–3
 - modulen `pylab`, 2
 - ones, 5
 - oppstart, 1
 - outer, 9
 - pcolor, 13
 - plot, 8
 - plotting av flater, 9–11
 - plotting av konturlinjer, 11–13
 - plotting av strømlinjer, 15
 - plotting av vektorfelt, 13–15

- print, 3
- py-filer, 6–7, 9, 11, 13
- quiver, 13–15
- range, 6
- set_xlabel, set_ylabel, set_zlabel,
11
- shape, 4
- size, 4
- sum, 4
- transpose, 3
- utskrift av grafikk til fil, 3
- zeros, 5

skalarfelt, 15

- hvordan beregne med Python, 9–11

strømfunksjon, 15

strømlinjer

- hvordan plote med Python, 15

trykk

- lavtrykk
 - i en lufttrykksmodell, 12
- lufttrykk
 - en modell som skal plottes, 11

trykkfelt, 15

vektorfelt

- hvordan plote med Python, 13–15