# Funksjoner med flere variable

*En prøveforelesning av Ottar Osen [(min side). (https://www.ntnu.no/ansatte/ottar.osen)](https://www.ntnu.no/ansatte/ottar.osen)*

Forelesningen varer 45 min og tar for seg: hva funksjoner er, funksjoner av flere variable, visualisering av funksjoner med flere variable, litt om Python programmering, øving på PC og til slutt en liten evaluering av forelesningen.

## Om funksjoner

Vi husker fra tidligere at når vi skriver $y = f(x)$ så er det slik at for én verdi av $x$ (i definisjonsmengden) så er det **én** verdi av $y$. Definisjonsmengden er jo alle lovlige verdier av $x$, ulovlige verdier er f.eks $x$-verdier som resulterer divisjon med 0 (når det er x i nevneren) eller fordi en tar kvadratrot av et negativt tall. F.eks. $f(x) = \frac{1}{x-2}$ som er ikke definert for $x = 2$, dvs $\mathscr{D}(f) = \mathbb{R}\backslash\{2\}$. Det viktige her er at det ikke finnes flere enn **én** $y$ verdi for én $x$-verdi.

Det betyr at f.eks. en sirkel **ikke** er en funksjon fordi der er det to $y$-verdier for hver $x$ verdi. Tegn en sirkel og overbevis deg selv om at så er tilfelle (og ja, det er 2 $x$-verdier som har bare en $y$-verdi). En sirkel i origo med radius 1 kan beskrives med ligningen $x^2 + y^2 = 1$, men for å få en funksjon så må vi nøye oss med en halv sirkel, da kan den skrives som $y = \sqrt{1 - x^2}, x \in [-1, 1]$

## Flere variable

I virkeligheten er det veldig ofte slik at det er mer enn én variabel i en funksjon. F.eks. har et rektangel arealet $A = l \cdot b$ og volumet av et prisme er $V = l \cdot b \cdot h$ for å vise at disse har heholdsvis to og tre variable kan vi gjerne skrive dem slik: $A(l, b) = l \cdot b$ og $V(l, b, h) = l \cdot b \cdot h$, da går det tydelig frem hvor mange variable vi har.

Fra fysikken kjenner vi mange formler som har flere variable. Disse kan gjerne skrives som funksjoner, f.eks. vil trykket på et legeme nedsenket i havet (f.eks. deg) være gitt av funksjonen $P(h, \rho, g) = \rho \cdot g \cdot h$ hvor $\rho$ er massetettheten (densitet), $g$ er tyngdens akselerasjon og $h$ er dybden. Nå vil du kanskje protestere og si at $\rho$ og $g$ er konstanter, men i virkeligheten er $\rho$ avhengig av bla. temperatur og saltinnhold [(ref (http://www.ric.edu/faculty/PSCI103/Seawater/Seawater_notes.htm))](http://www.ric.edu/faculty/PSCI103/Seawater/Seawater_notes.htm), dvs. $\rho(T, S)$ og $g$ avhenger av hvor på jorda du er, dvs, $g(Lat, Long, z)$ [(ref (https://snl.no/tyngdens_akselerasjon))](https://snl.no/tyngdens_akselerasjon).

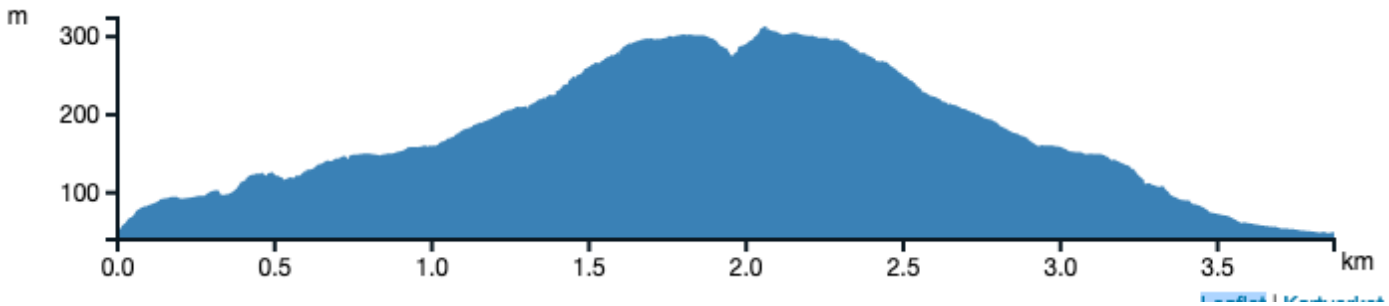
Et eksempel på 4 variable kan være en funksjon som beskriver temperaturen på et nysteikt brød som du akkurat har tatt ut av ovnen. Det er lett å se at temperaturen avhenger av hvor du måler på brødet og hvor lenge det har gått siden du tok det ut av ovnen: $T(x, y, z, t)$. Her er $x$, $y$, og $z$ koordinatene i rommet og $t$ er tiden som har gått.

# Visualisering av funksjoner med flere variable

På papiret kan vi enkelt tegne funksjoner med én variabel, når vi har to variable så blir det litt vanskeligere. En funksjon med én variabel $y = f(x)$ gir oss punkter $(x_0, y_0), (x_1, y_1), (x_2, y_2), \ldots . (x_n, y_n)$ som vi lett kan plassere i et koordinatsystem på et flatt papir (en flate). Papiret har altså 2 dimensjoner noe som er perfekt for en variabel. Derimot så vil en funksjon med to variable $z = f(x, y)$ gi oss punkter $(x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y_2, z_2), \ldots . (x_n, y_n, z_n)$ som dermed må plottes i et koordinatsystem med tre akser. Generelt gjelder det derfor at hvis du har $n$ variable så vil grafen være en $n$ dimensjonal flate ($\mathbb{R}^n$) i et $n + 1$ dimensjonalt ($\mathbb{R}^{n+1}$) rom (koordinatsystemet har $n + 1$ akser). Allerede ved 2 variable ($n = 2$) får vi en utfordring fordi da trenger vi et tredimensjoanlt rom for å tegne et plan. Det er ikke så lett å få til på et flatt papir!

En måte å vise 3 dimensjoner på et plan er å vise bare en projeksjon fra 3 ned til 2 dimensjoner. En slik løsning som bør være velkjent er turkart. Her er projeksjonen langs $z$-aksen og ned i $x - y$ planet. Høyden er vist ved hjelp av kote-linjer:

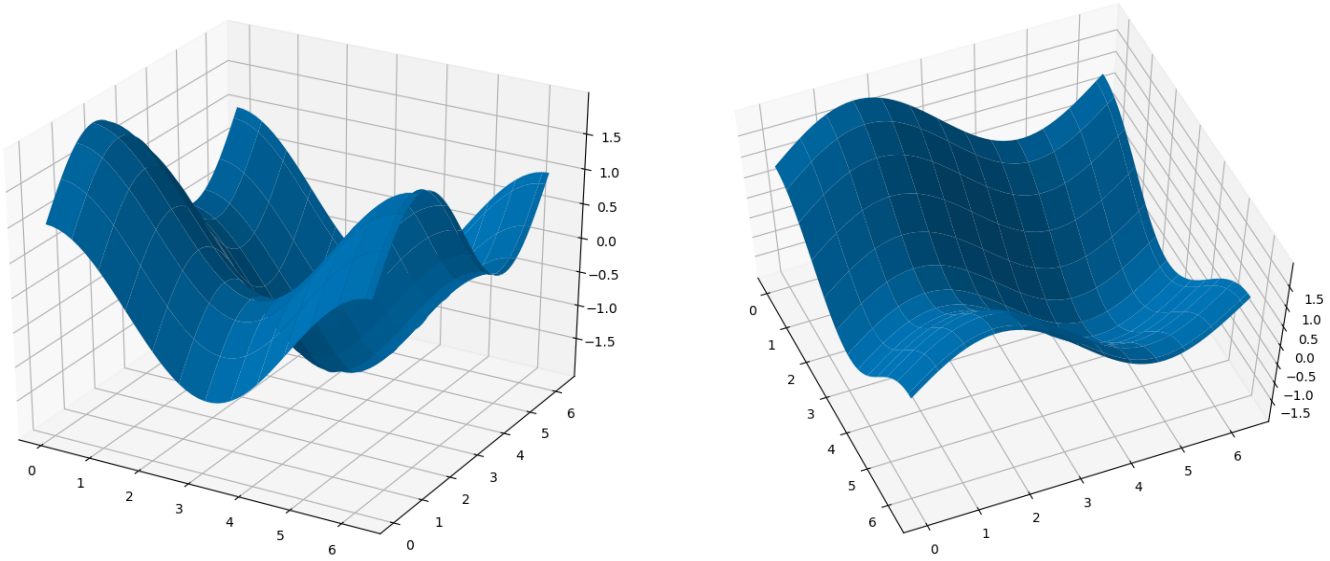

Men for å få et godt bilde av høydevariasjonen langs løypa (svart) vil en annen projeksjon være bedre, f.eks. en langs $x$-aksen og inn i $y - z$ planet. Da kan en få en høydeprofil lignende som denne:




Ved å se på avstanden mellom kotene ser vi hvor bratt det er. På dette kartet er det ikke vist noen elver, men hvor vil vannet renne? Øvelse: gå til [www.norgeskart.no (https:www.norgeskart.no)](https:www.norgeskart.no) og sjekk vinkelen mellom elver og kvotelinjer, hva ser du?

Vi legger også merke til at det i kartet også ligger annen informasjon i form av farge. En mulighet er å la fargen vise funksjonsverdien. Et eksempel på dette er et termografibilde:


Det er ganske vanlig å vise 3 dimensjonale figurer som en projeksjon sett fra et observasjonspunkt som ikke ligger på noen av aksene slik at vi får et perspektiv. Avhengig av geometrien på en slik figur kan en slik løsning skjule deler av flaten og det kan være vanskelig å lese av noenlunde riktige verdier langs aksene. På en datamskin har vi fordelen av å kunne rotere koordinatsystemet (flytte observasjonspunktet) og zoome inn og ut slik at en kan studere figuren fra forskjellige sider:



Samme graf vist fra 2 ulike observasjonspunkt.

Slike figurer kan lages på PCen med en lang rekke program som f.eks. Matlab, Dpgraph, Derive, Matematica, Maple, MathCad og Geogebra. I det neste kapitlet vil vi se på et populært programmeringsspråk som er meget velegnet til både beregninger og visualisering.

# Python

Phyton er et programmeringsspråk som er blitt veldig populært de siste årene. P.g.a. gode støtte for matematikk og statistikk er språket blitt veldig populært i.f.m. data analyse og kunstig intelligens. Python har utvidelser som gir bedre støtte for beregninger og plotting. De mest populære utvidelsene er Numpy (matematikk) og Matplotlib (figurer ala Matlab).

Dette dokumentet du leser nå er laget med et verktøy som heter Jupyter. Jupyter lager notatbøker som kan inneholde blanding av tekst, figurer, matematiske ligninger osv. og med støtte for programmering inne i selve dokumentet! Jupyter støtter **mange** programmeringsspråk, men det mest populære er Python.

*Dette avsnittet er bare for de som er nyskjerrige på Python og Jupyter. Hvis du har lyst til å lage dine egne Jupyter notatbøker så finnes det flere gratis ressurser på nettet. Selv bruker jeg Azure sin gratis tjeneste Azure Notebooks (https://notebooks.azure.com/). Her finner du også en introduksjon til Python (https://notebooks.azure.com/Microsoft/projects/2018-Intro-Python/html/Introduction%20to%20Python.ipynb). W3Schools.com har et fint interaktivt kurs i Python (https://www.w3schools.com/python/) og **mange** andre programmeringsspråk. Felles for ressursene over er at du ikke behøver å installere noe på din egen PC, alt du trenger er en nettleser! Hvis du etterhvert vil lære deg mer Python anbefaler jeg å installere Python på din egen PC. En av de mest populære distribusjonene av Python heter Anaconda og kan lastes ned her (https://www.anaconda.com/distribution/), den finnes både for Windows, MacOS og Linux.*

Men, for å kjøre programmene nedenfor behøver du **ikke** å foreta deg noe av det som er beskrevet over.

I boksen nedenfor importerer jeg de utvidelsene vi trenger. Det er tilstrekkelig å ha disse med en gang på en side.

In [1]:
```
from numpy import *
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D
```

Da er vi klare til å plotte figurer. I eksemplene nedenfor har jeg laget 2 og 2 bokser, en som inneholder programmet som lager og bruker funksjonen for å skape dataene og en som foretar selve plottingen av dataene.

Nederst i dokumentet finner du flere tomme bokser som du kan bruke til å lage dine egne program!

Først skal vi se på et par eksempler.

## Eksempel 1

Vi skal tegne grafen for funksjonen:

$$f(\phi_m, \phi_p) = 2 + \alpha - 2\cos(\phi_p)\cos(\phi_m) - \alpha\cos(\phi_{ext} - 2\phi_p)$$

hvor,

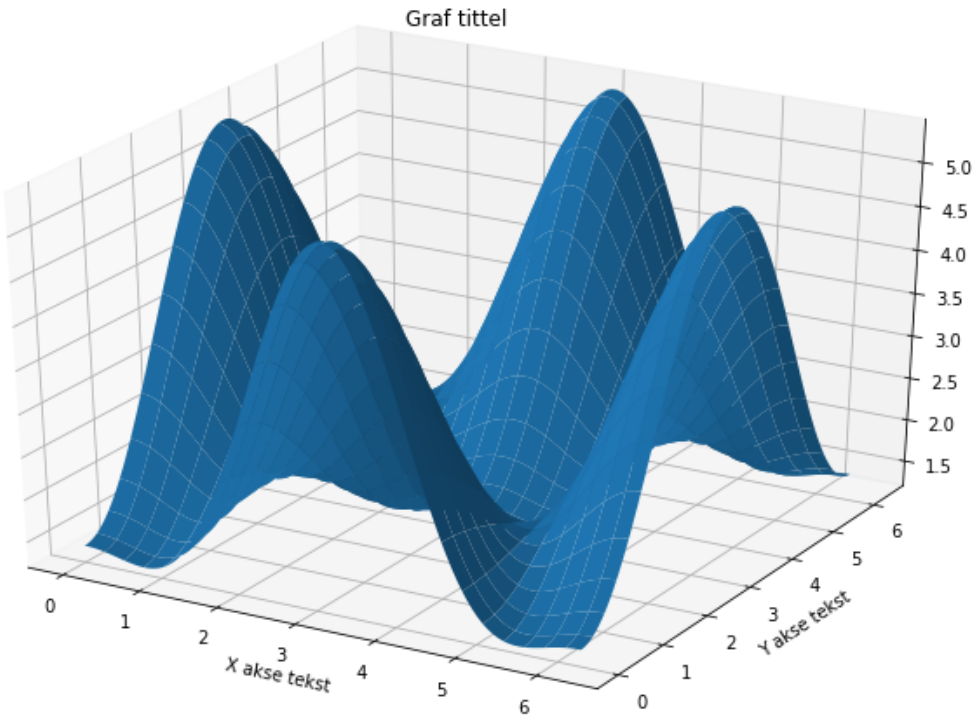
$\phi_m$ og $\phi_p$ er i intervallet $[0, 2\pi]$ og $\alpha = 0.7$ og $\phi_{ext} = 2\pi \cdot 0.5$

In [2]:
```
alpha = 0.7 # konstanter
phi_ext = 2 * pi * 0.5

def f(phi_m, phi_p): # definisjon av selve funksjonen
    return 2 + alpha - 2 * cos(phi_p)*cos(phi_m) - alpha * cos(phi_ext - 2*phi_p)

phi_m = linspace(0, 2*pi, 100) # lager 100 verdier fra 0 til 2*pi
phi_p = linspace(0, 2*pi, 100)
X,Y = meshgrid(phi_p, phi_m) # Lager alle kobinasjoner av x og y
Z = f(X, Y).T # Kaller selve funksjonen og legger resultatet i Z, .T er
 transpose, dvs. bytter om kolonner og rader.
```

In [3]:
```
## Magic kommandoen på neste linje gir en figur som ikke kan roteres.
%matplotlib inline
fig = plt.figure(figsize=(12,8)) # lager en tom figur
ax = plt.axes(projection='3d') # legger inn 3 akser
plt.xlabel('X akse tekst')
plt.ylabel('Y akse tekst')
plt.title('Graf tittel')
plt.grid(True)
p = ax.plot_surface(X, Y, Z, rstride=4, cstride=4, linewidth=0) # plotte
r dataene
```



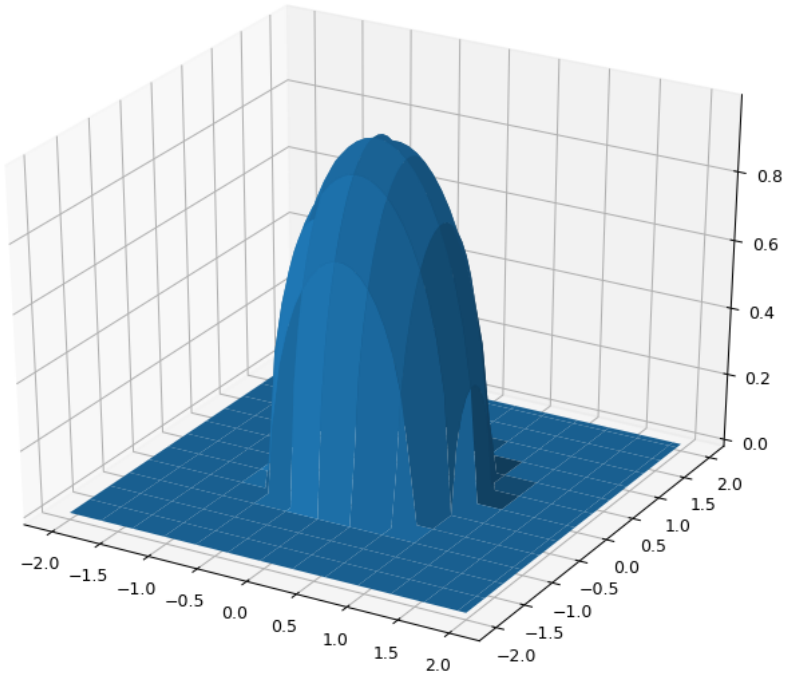

## Eksempel 2

Vi ønsker å plotte funksjonen $f(x, y) = \sin(x) + \cos(y)$ , hvor $x \in [0, 2\pi]$ og $y \in [0, 2\pi]$

In [4]:
```
def min_func(x, y):
    return sin(x) + cos(y)

x = linspace(0, 2*pi, 50)
y = linspace(0, 2*pi, 50)
X,Y = meshgrid(x, y)
Z = min_func(X, Y).T
```

In [10]:
```
## Magic kommandoen på neste linje gir en figur som KAN roteres.
%matplotlib notebook
fig = plt.figure(figsize=(10,8))
ax = plt.axes(projection='3d')
p = ax.plot_surface(X, Y, Z, rstride=4, cstride=4, linewidth=2)
```



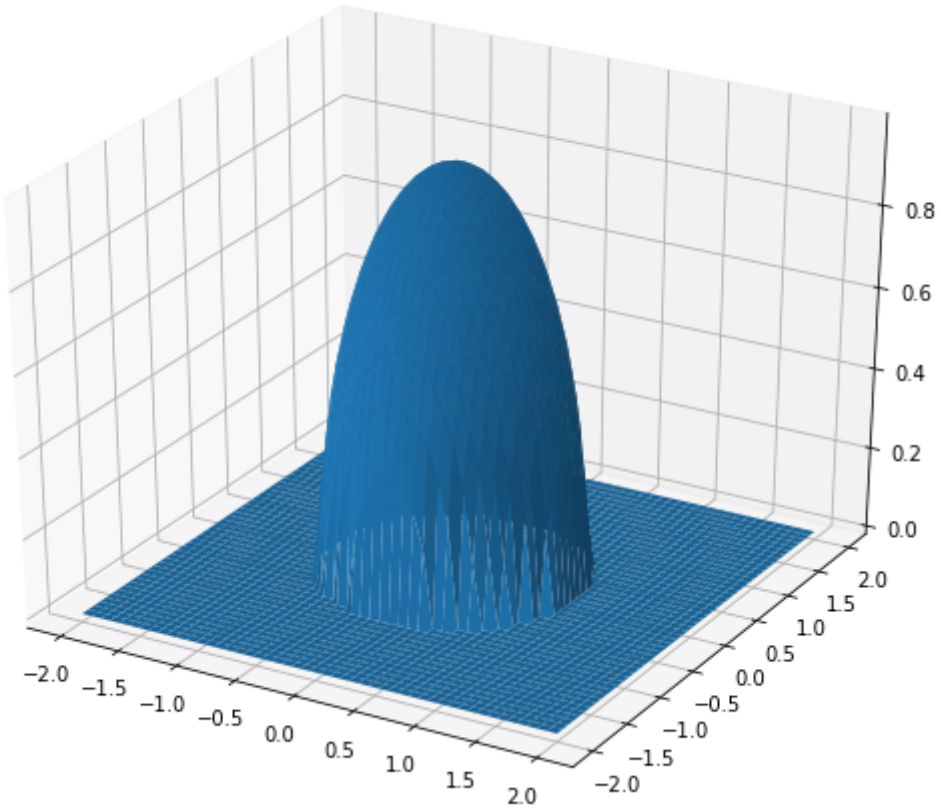

## Eksempel 3

Vi ønsker å plotte funksjonen $f(x, y) = \sqrt{1 - x^2 - y^2}$. Her er det ikke så lett å se hva som blir definisjonsmengden, men nedenfor ser du et lite triks i Ludo. Vi legger uttrykket under rottegnet i en variabel $tr$ (vektor). Deretter lager jeg en ny variabel $nu$ (vektor) som består av 0 eller 1. 0 hvis tilsvarende verdi i $tr$ er negativ. Avslutningsvis ganger jeg $tr$ med $nu$ og gjør dermed alle negative verdier i $tr$ til 0. Til slutt tar jeg kvadratroten av $tr$ som nå er en vektor med bare ikke-negative tall.

In [6]:
```
def min_func(x, y):
    tr = 1-x**2-y**2
    nu = tr > 0
    tr = tr * nu
    return sqrt(tr)

x = linspace(-2, 2, 50)
y = linspace(-2, 2, 50)
X,Y = meshgrid(x, y)
Z = min_func(X, Y).T
```

In [7]:
```
## Magic kommandoen på neste linje gir en figur som IKKE kan roteres.
%matplotlib inline
fig = plt.figure(figsize=(10,8))
ax = plt.axes(projection='3d')
p = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, linewidth=2)
```




Du kan gjerne endre på eksemplene over eller du kan kopiere koden inn i cellene nedenfor. I cellene kan du legge inn program eller regnestykker. Trenger du flere celler så merk en celle ved å klikke til venstre slik at venstre kant av ramma blir blå. Trykk så på B (for Below). For å få bereknet cella/kjørt programmet; trykk på Shift + Enter. Husk at $2^3$ skrives som 2**3 (** ikke ^ som i andre programmeringsspråk). Kvadratrot skrives som sqrt( ), eller du kan jo opphøye i 0.5.

In [8]:
```
5*sqrt(100/7)/4**1.29
```

Out[8]:
```
3.160556769807855
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: