KØBENHAVNS UNIVERSITET

Optimering

Programmering og Modellering

Katrine Hommelhoff Jensen

Hvad er optimering?

Mange vil sige *kernen af datalogi* - det handler om at finde den *bedste* d.v.s. billigste/hurtigste/korteste/mindste løsning, f.eks.:

- Navigationssystemer: hurtigste og/eller korteste vej, givet diverse begrænsninger
- Ugeskema-planlægning: mindste overlap mellem lærere
- Planlægning af offentlig trafik: servicering af flest borgere med mindst mulig...service?
- Transportpakning: flest pakker i lastbil
- Vindmøller: den optimale form af vingen

Programmer der finder disse bedste løsninger kalder vi for *optimeringsalgoritmer*

Hvad er optimering?

At finde den bedste løsning til et problem kan oftest oversættes til at minimere eller maksimere en funktion, dvs. finde det x^* for hvilket $f(x^*)$ er minimum eller maksimum

- At minimere en funktion $f(x)$ skrives typisk $\min_x f(x)$ dvs. "minimering over x"
- Maksimering og minimering er reelt det samme problem, dvs. $\max_x f(x) = \min_x -f(x)$, derfor tales der typisk bare om minimering
- f kaldes typisk *objektivfunktionen*

Mange *kategorier* af optimeringsalgoritmer, hver med sin *strategi*, styrker og svagheder

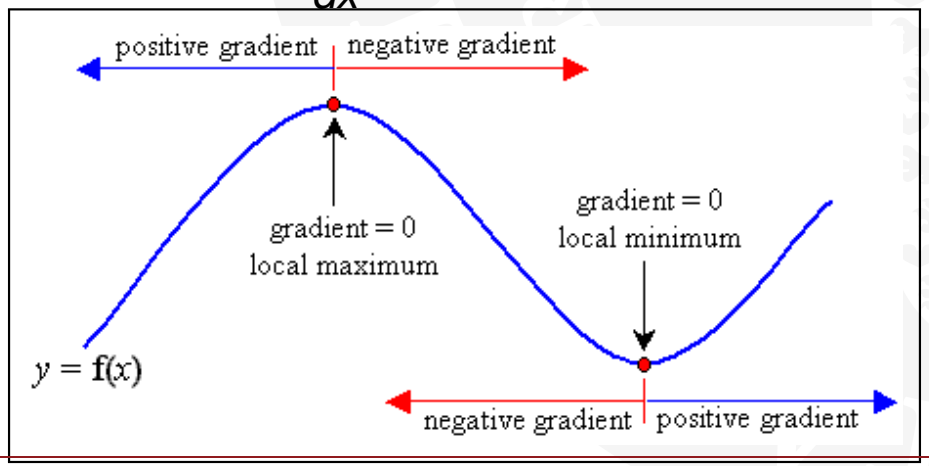
Minimering af en funktion

Vi starter ved det kendte...

- Minimum (eller maksimum) er kendetegnet ved at den afledte funktion (hældningen) er nul
- Minimum (eller maksimum) af $f(x)$ er betinget af at den afledte $f'(x) = 0$, også skrevet $\frac{df(x)}{dx} = 0$
- Findes et x for hvilket $f'(x) = 0$, så afgører den dobbelafledte $f''(x)$ om der er tale om et minimum eller et maksimum:

  $f''(x) < 0$ også skrevet $\frac{d^2 f(x)}{dx^2} < 0 \quad \rightarrow$ maksimum

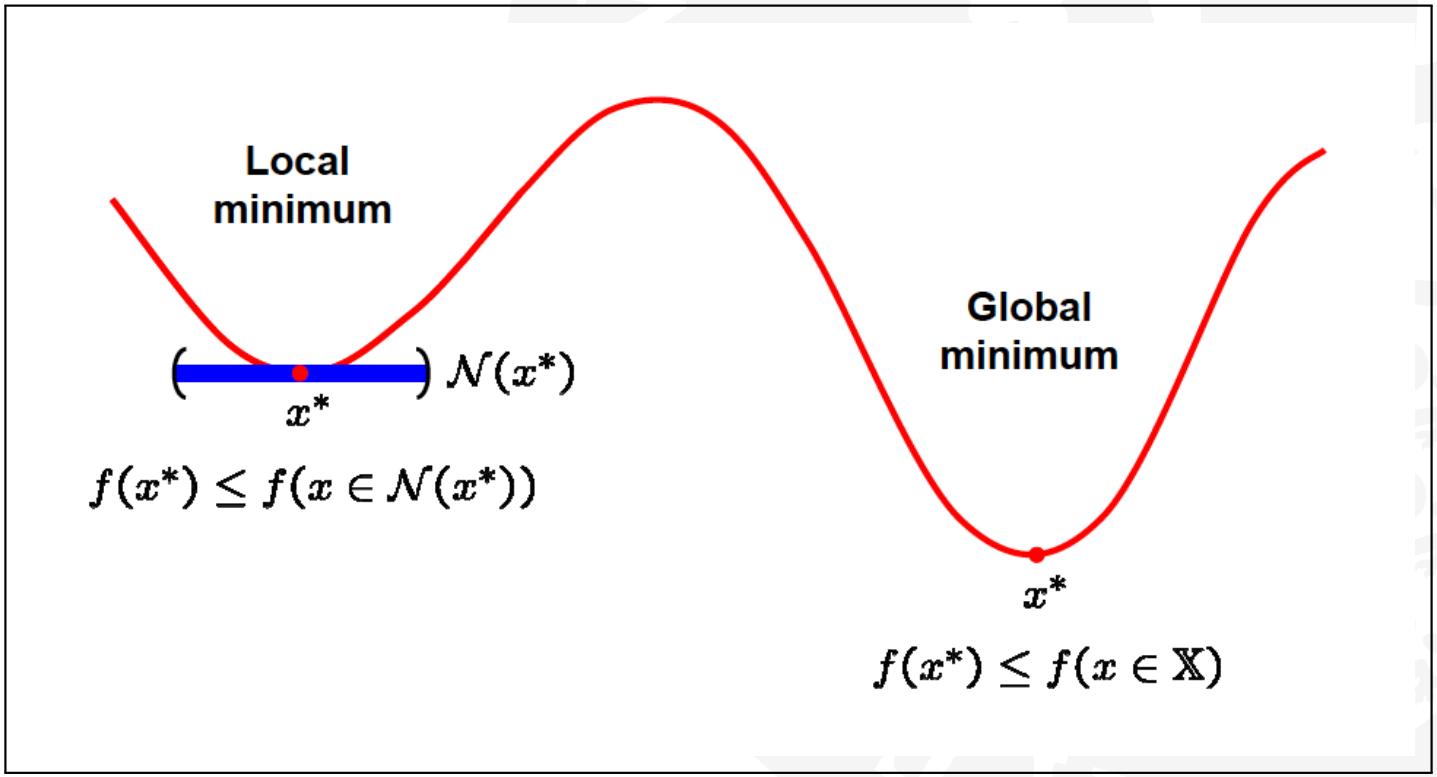
  $f''(x) > 0$ også skrevet $\frac{d^2 f(x)}{dx^2} > 0 \quad \rightarrow$ minimum



Hvad er et minimum egentlig?

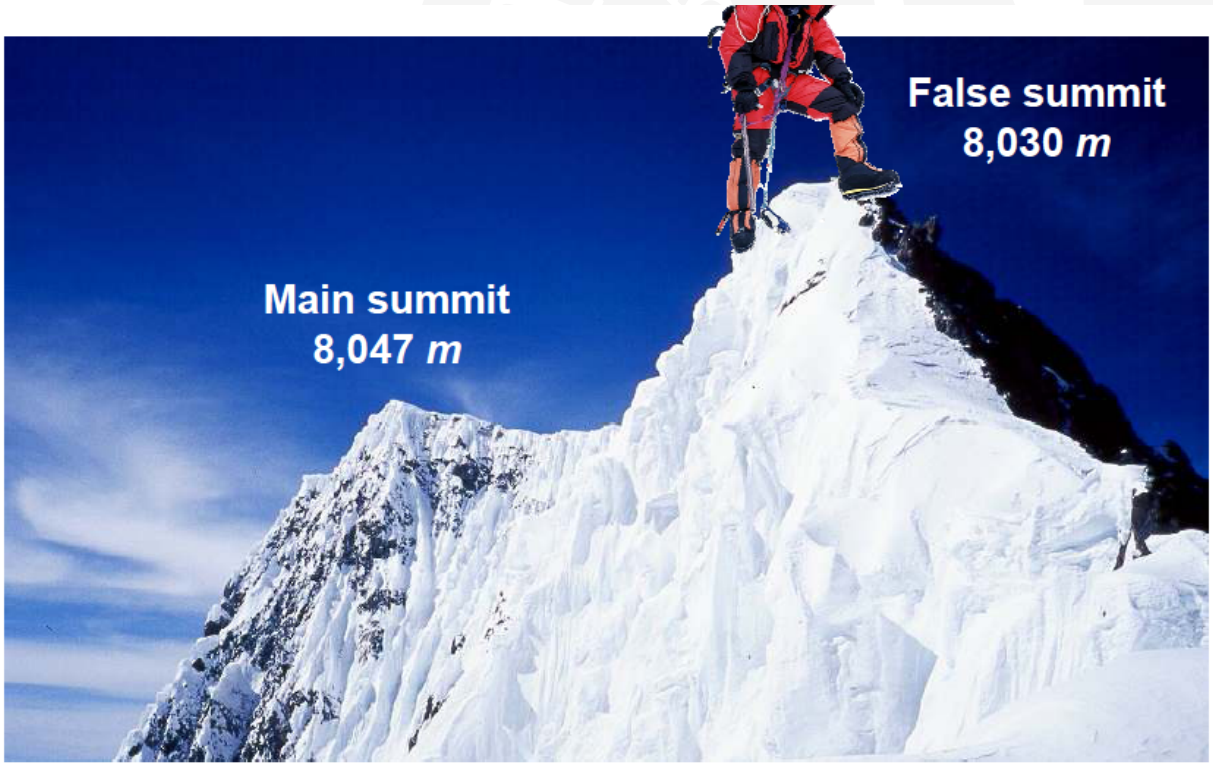
Lokal vs. globalt minimum - optimeringsmetoden skal vælges derefter

- Lokalt minimum som oftest trivielt
- Globalt minimum kan være omvendt svært at finde



Hvad er et minimum egentlig?

Typisk kan et minimum først afgøres som lokalt eller *måske* globalt, når man er nået frem til det. Derefter stadig ingen garanti for globalt minimum.

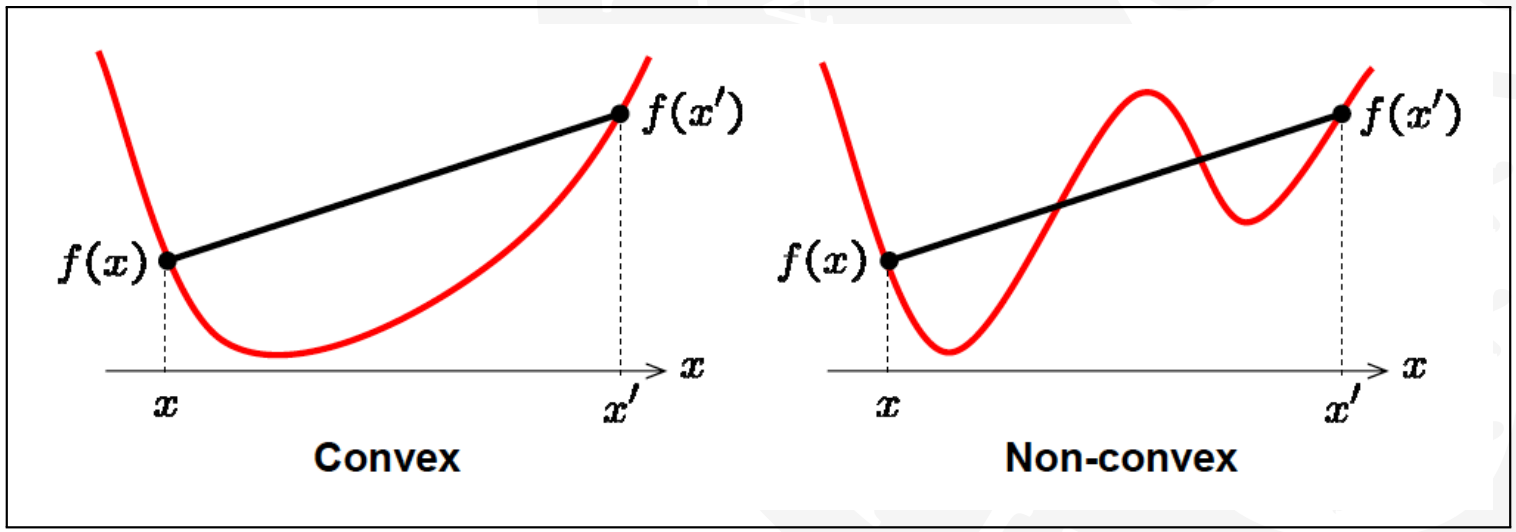


Broad Peak (K3), 12th highest mountain on Earth

Hvad er et minimum egentlig?

Konveksitet af objektivfunktion - eneste egenskab der kan garantere konvergens mod globalt minimum. For en konveks funktion gælder:

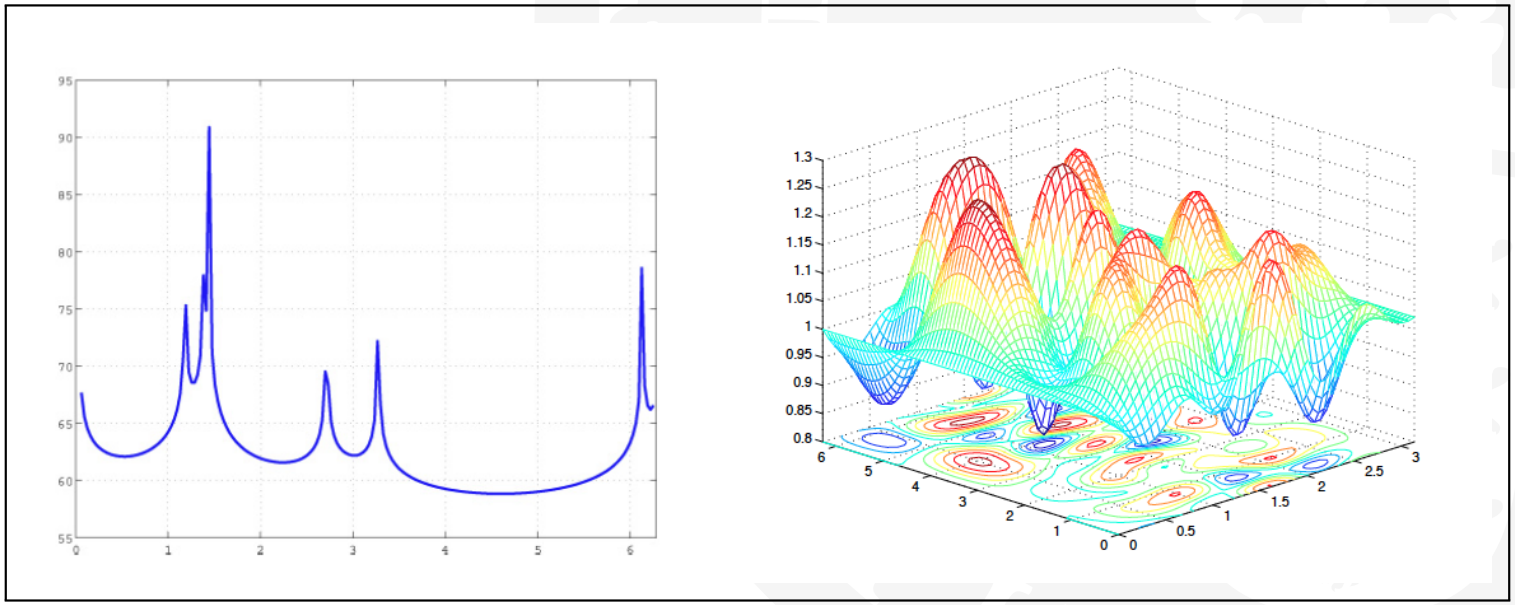
- mellem ethvert par af punkter på funktionen kan spændes en linie som ikke krydser funtionen
- lokalt minimum = globalt minimum



n-dimensionelle funktioner

For en funktion $f(\mathbf{x})$ for $\mathbf{x} \in \mathbb{R}^n$ gælder

- de samme problematikker, dog meget mere kompliceret (tidskrævende) at minimere
- de mest generelle optimeringsalgoritmerne *søger* derfor efter minima, ingen direkte løsning



Hældning, 1. ordensafledte og gradient

Vores vigtigste navigationsredskab til søgning efter minimum

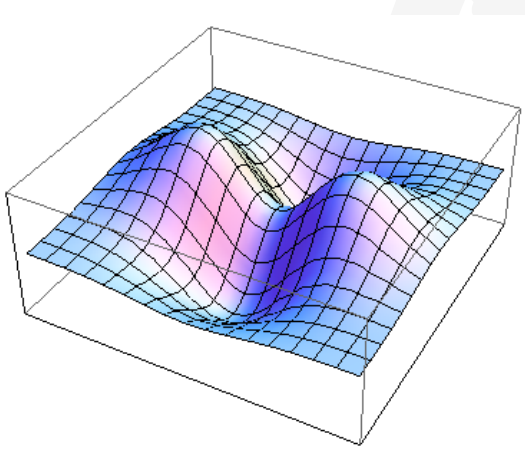
- Gradienten af f i x betegnes $\nabla f(x)$
- I $\mathbb{R}$ er gradienten $\nabla f(x) = f'(x)$
- I $\mathbb{R}^n$ dvs. for n-parameter funktioner $\nabla f(\mathbf{x})$ hvor $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ er gradienten udgjort af de førsteordens partielt afledte

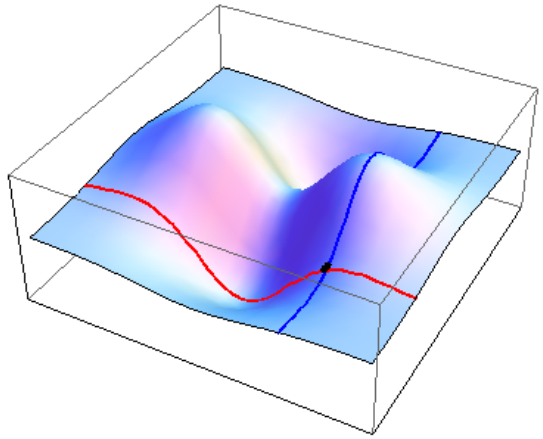
$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}(\mathbf{x}), \frac{\partial f}{\partial x_2}(\mathbf{x}), \ldots, \frac{\partial f}{\partial x_n}(\mathbf{x}) \right)$$

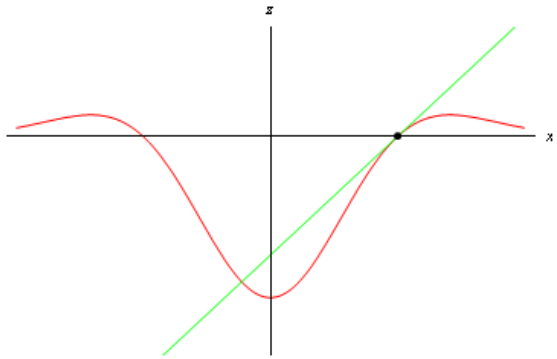
- Gradienten $\nabla f(\mathbf{x})$ peger i retningen af den maksimale stigning af f i punktet $\mathbf{x}$

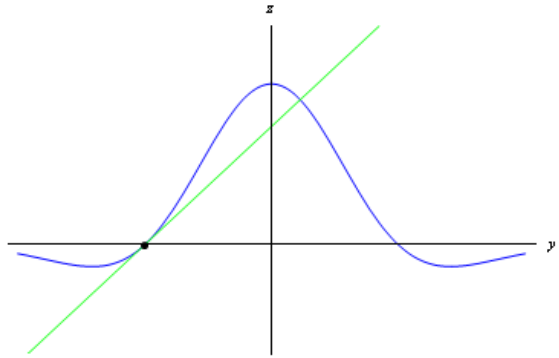
Hældning, 1. ordensafledte og gradient

De førsteordens partielt afledte angiver hældningen i et punkt m.h.t. hver dimension af rummet, f.eks. $\mathbb{R}^2$:



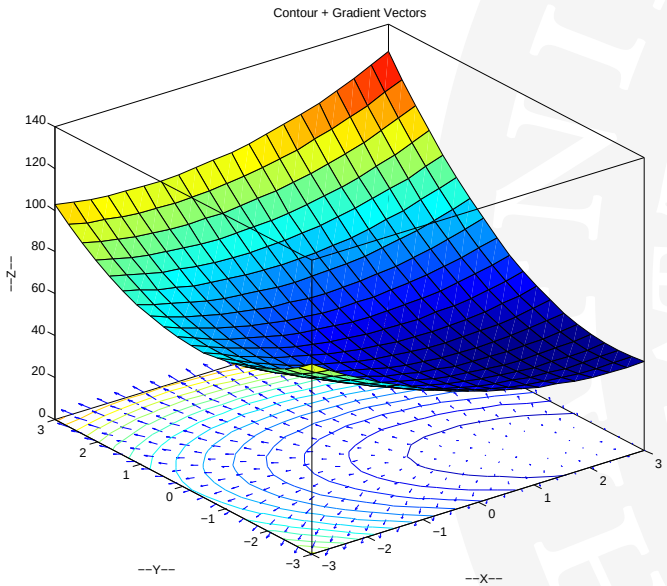


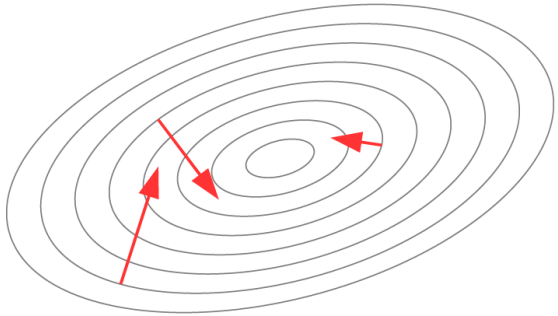




Hældning, 1. ordensafledte og gradient

Gradienten angiver retningen for den *største stigning* i et punkt, svarende til den retningsafledte:





Konturplot illustrerer hvordan vi kan bruge gradienten til at navigere i den modsatte retning - mod et minimum

Krumning, 2. ordensafledte og Hessian

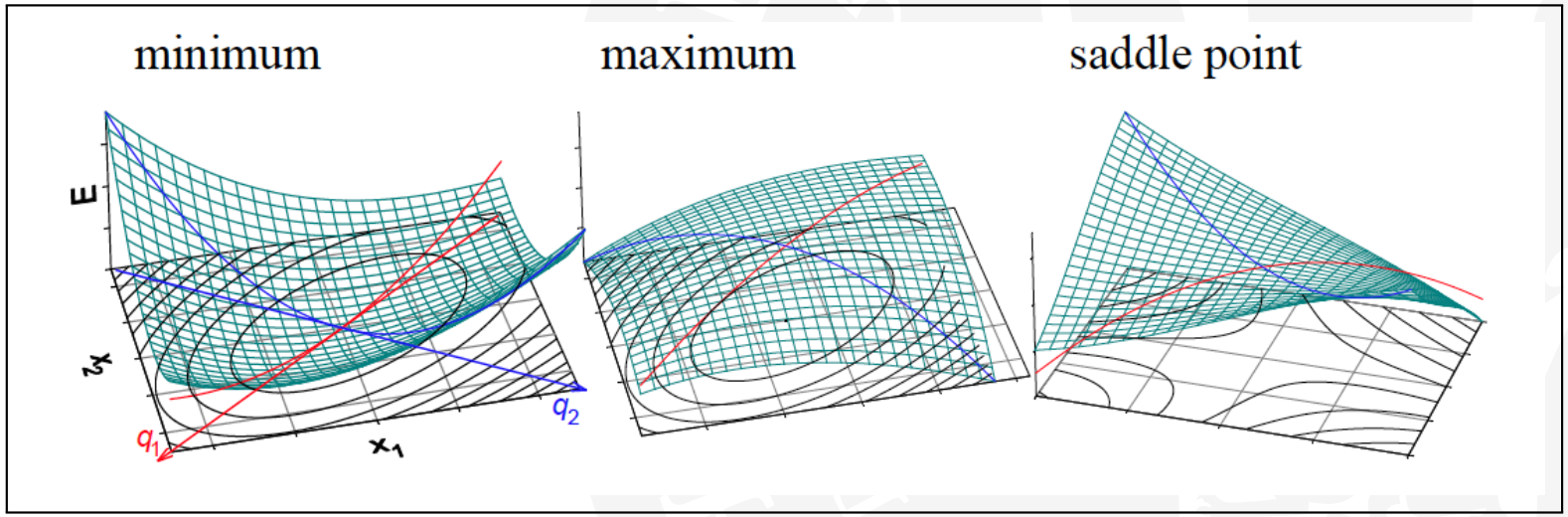
Et andet vigtigt (men dyrt) navigationsredskab

- 1. ordensafledte giver *lineær* lokal approximation, 2. ordensafledte giver *kvadratisk* lokal approksimation
- Ligesom gradient indeholder de førsteordens partielt afledte i et punkt, indeholder *Hessian matricen* de andenordens partielt afledte
- Hessian matricen for $f(\mathbf{x})$ er givet ved

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Krumning, 2. ordensafledte og Hessian

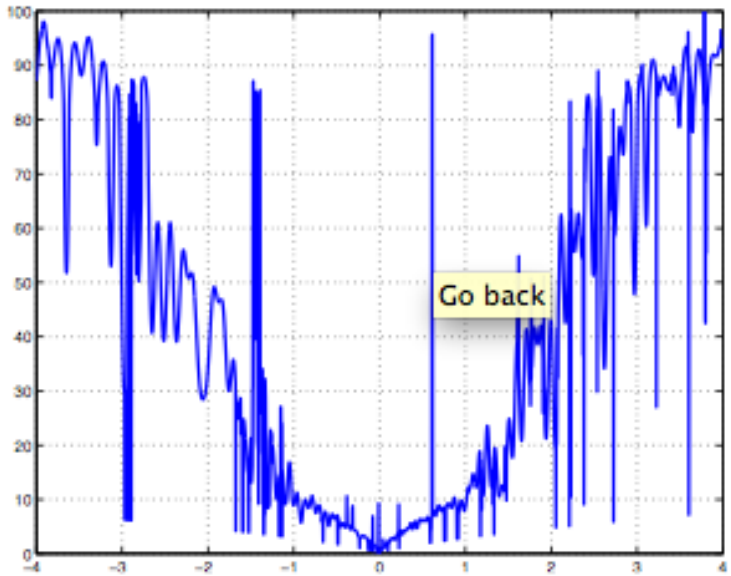
Hessian matricen kan fortælle os om lokalområdet er et *minimum*, *maximum* eller *saddelpunkt*



De underliggende konturplot bliver ofte brugt til at illustrere de enkelte trin i optimeringsalgoritmerne

Hvad gør en funktion svær at minimere

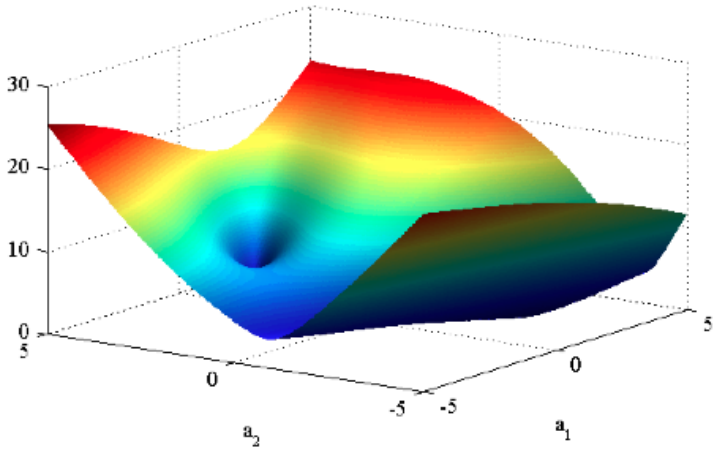
Vi har flere udfordringer end n-dimensionalitet, der kan afgøre valg af optimeringsalgoritme, f.eks. hvis funktionen er *ikke-glat*, *diskontinuert* og/eller *støjfyldt*:

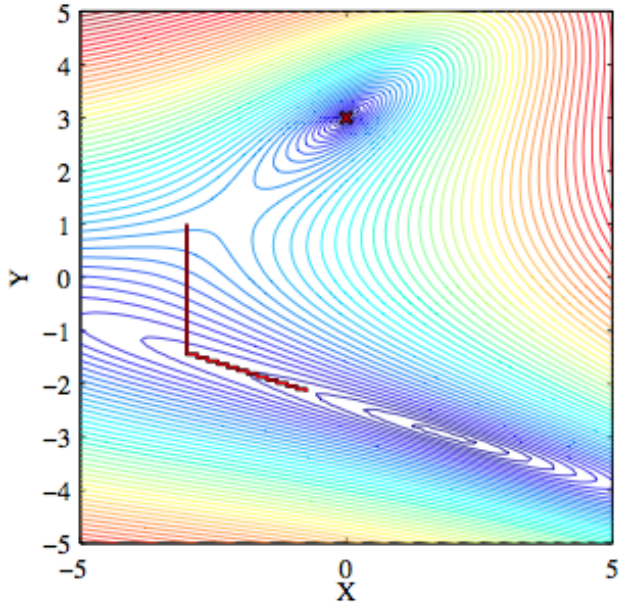


Dette repræsenterer desværre meget godt data indsamlet fra den *virkelige verden* - derfor skal vi bl.a. snakke om måder at minimere *diskret data*

Hvad gør en funktion svær at minimere

De søgende optimeringsmetoder kan også blive udfordret af specielle typer af landskaber, udover mange lokale minima - *sammentrykkede gradienter*:



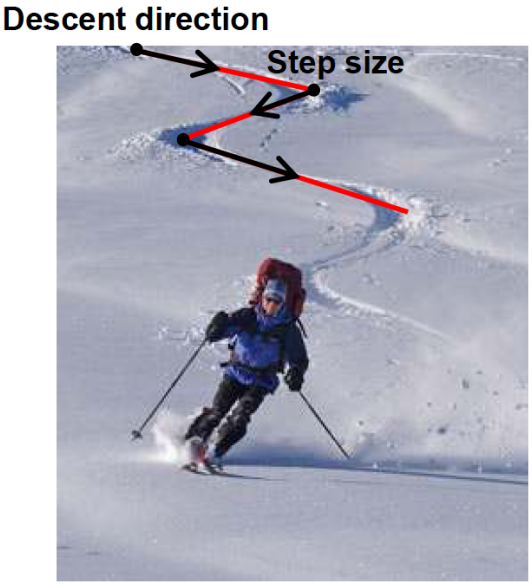


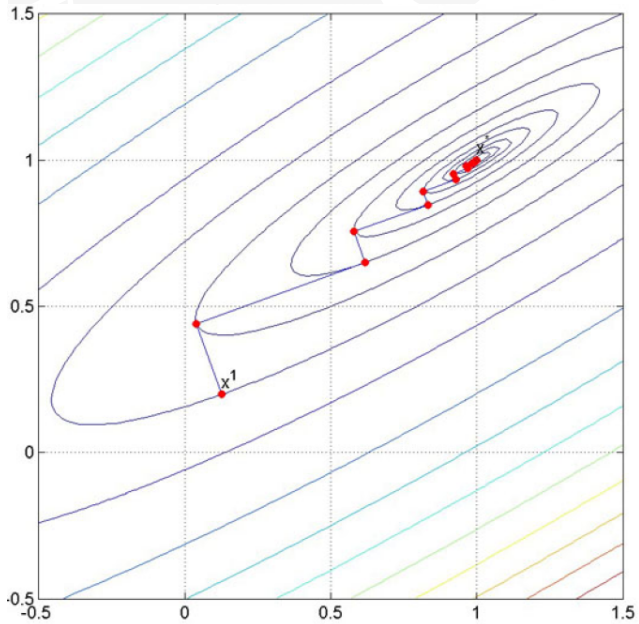
Hvordan finder vi minima her?

Optimeringsalgoritmer

Lad os få en mere formel definition

- Iterative procedurer der for en funktion $f(\mathbf{x})$ genererer en sekvens $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n-1}, \mathbf{x}_n)$ af parameterværdier, hvor, ved tidsskridt n er $\mathbf{x}_n$ et estimat af optimum $\mathbf{x}^*$
- Konvergens afgøres typisk ved een af to måder:
  - $|\mathbf{x}_n - \mathbf{x}^*| = 0$ (urealistisk)
  - Givet en præcision ε, $\mathbf{x}_n$ er optimum når $|\mathbf{x}_n - \mathbf{x}^*| \leq \varepsilon$





Generisk optimeringsmetode

Generelt princip

1. Vælg et startpunkt (parameter) $\mathbf{x}_k$, $k = 1$
2. Så længe vi ikke har fundet et minimum, gør data:
   1. Vælg en *nedstigningsretning* $\mathbf{d}_k$
   2. Vælg en *skridtstørrelse* λ_k
   3. Sæt næste punkt $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k$
3. Løsningen er $\mathbf{x}^* \approx \mathbf{x}_k$

De enkelte descent metoder har hver deres bud på

- Hvordan vælges nedstingningsretningen?
- Hvordan vælges skridtstørrelsen - ER det størrelsen eller gørelsen? Tradeoff:
  - Små skridt: Tidskrævende
  - Store skridt: Omveje, zig-zag bevægelse mod optimum

Optimalitetsbetingelser

Hvornår konvergerer vi?

- Hvis $\mathbf{x}^*$ er et lokalt optimum af f, så er $\nabla f(\mathbf{x}^*) = 0$ og $\nabla^2 f(\mathbf{x}^*)$ er *positive semi-definite*...

Mere realistiske stopkriterier:

- Tæt på lokalt minimum $\nabla f(\mathbf{x}) \approx 0$, dvs. stop når gradienten bliver *lille nok*

$$\nabla f(\mathbf{x}_k) \leq \varepsilon \tag{1}$$

- Stop når ændringen i objektivfunktionen er lille nok

$$\frac{f(\mathbf{x}_k) - f(\mathbf{x}_{k+1})}{f(\mathbf{x}_k)} \leq \varepsilon \tag{2}$$

Typer af optimeringsmetoder

Deterministiske metoder

- Lokale
- Gradient-baserede

Stokastiske metoder

- Globale
- Baseret på tilfældige valg

Valget af hvilken optimeringsmetode afhængigt af problemets natur

- skal vi tilstræbe et globalt minimum eller er et lokalt godt nok?
- hvor kompliceret er vores funktion?

Gradient-baserede optimeringsalgoritmer

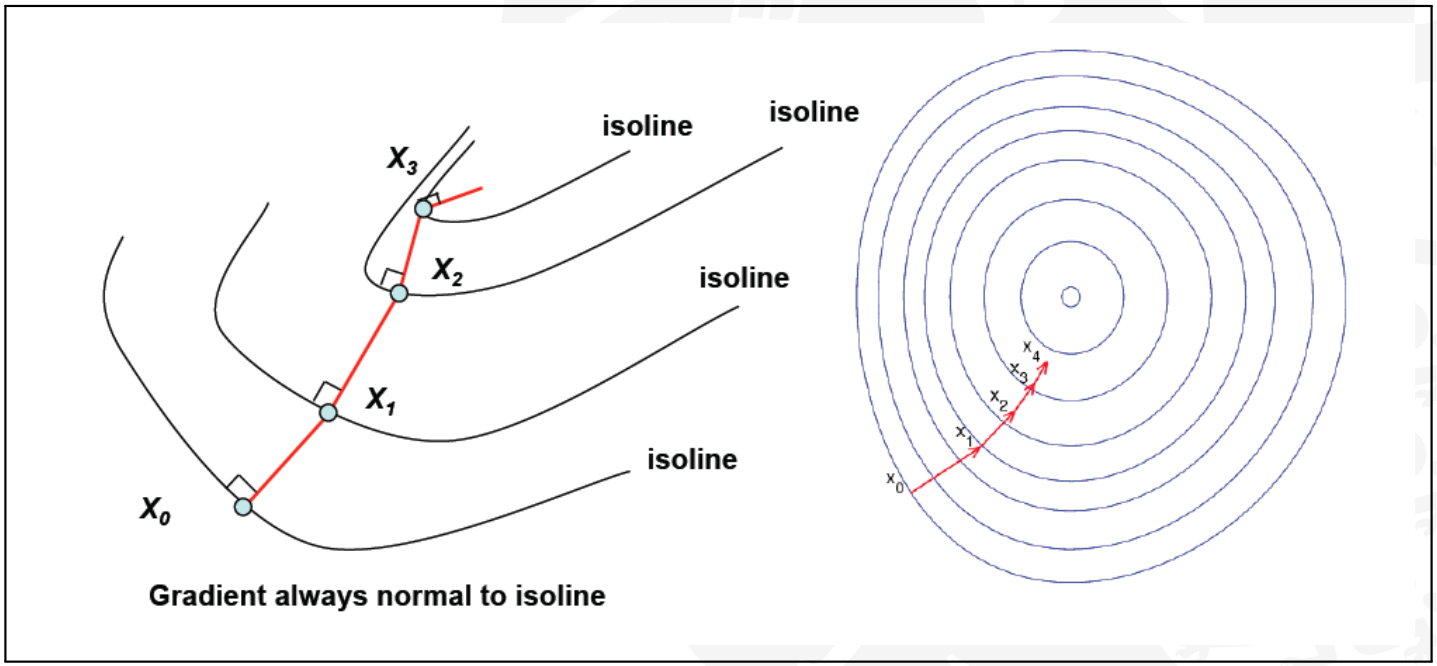
En hurtig opfrisker: Klassiske metoder til at løse $f'(x) = 0$:

- Newton-Raphson: Approksimerer f' med en lineær funktion g (vha. f'') og finder så roden af denne - hvilken ikke nødvendigvis er roden af f', men et godt gæt og konvergerer oftest hurtigere end gradient descent.
- Bisektion: Deler iterativt et interval og vælger det delinterval, hvori roden må findes

De gradientbaserede, deterministiske optimeringsmetoder anvendes på *n-dimensionelle* funktioner og anvender information om *gradienten* og måske også *Hessian* til at navigere

Gradient-baserede optimeringsalgoritmer

- Gradient descent
- Steepest descent
- Newton



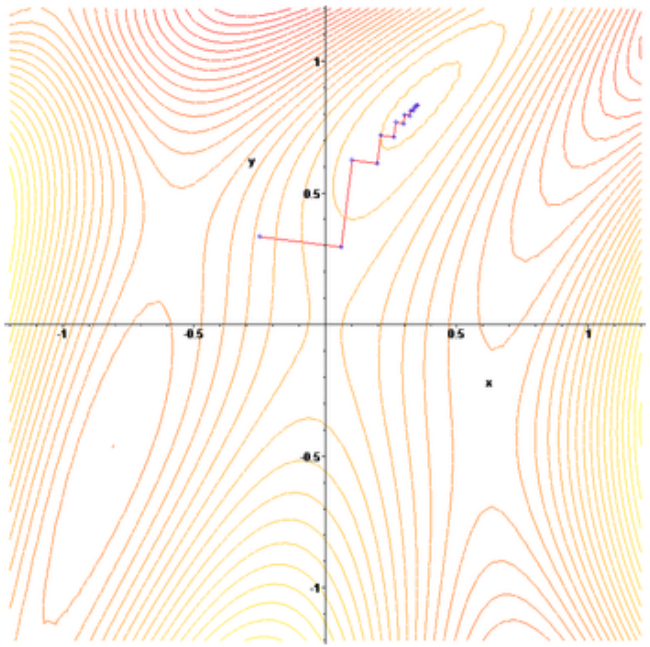
Konturplot: Gradienten navigerer mod lokalt minimum

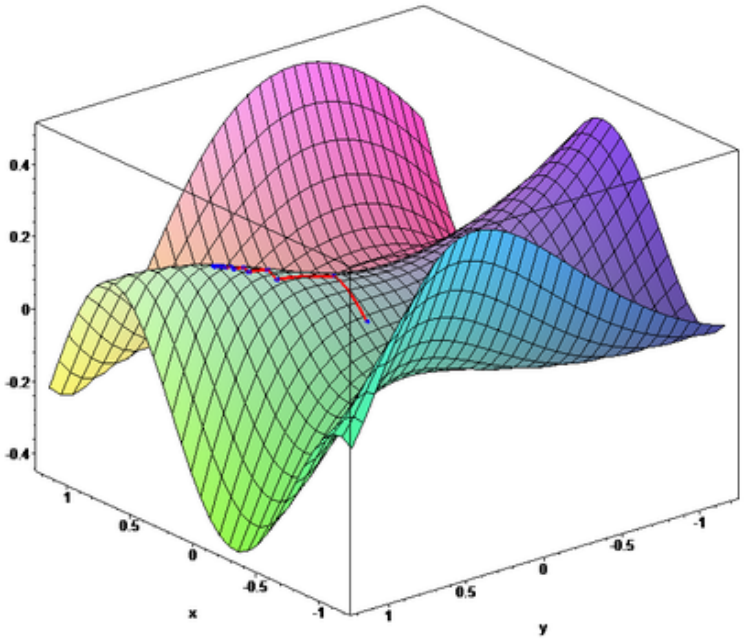
Gradient descent

- Nedstigningsretningen sættes til den negative gradient $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$ - dette kaldes også den stejleste nedstigningsretning (steepest descent direction)
- Vælg en skridtstørrelse, to overordnede muligheder:
  - Konstant skridtstørrelse - f.eks. beregnet udfra værdien af den maksimale afledte
  - Forskellig / adaptiv skridtstørrelse for hver iteration - f.eks. steepest descent metoden
- Metoden er 'dyr' hvis gradienten skal beregnes præcist

Gradient descent

$$F(x,y) = \sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)\cos(2x + 1 - e^y)$$





Illustrerer tydeligt gradient descent metodens 'lokalitet' - konvergerer hurtigt i et lokalt minima (men dog hurtigt)

Gradient descent

Lidt hurtig Python-kode

```
x_old = 0
x_new = 6 # The algorithm starts at x=6
tau = 0.01 # step size
precision = 0.00001

def f_prime(x):
    return 4 * x**3 - 9 * x**2

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - tau * f_prime(x_old)

print("Local minimum occurs at", x_new)
```

Steepest descent

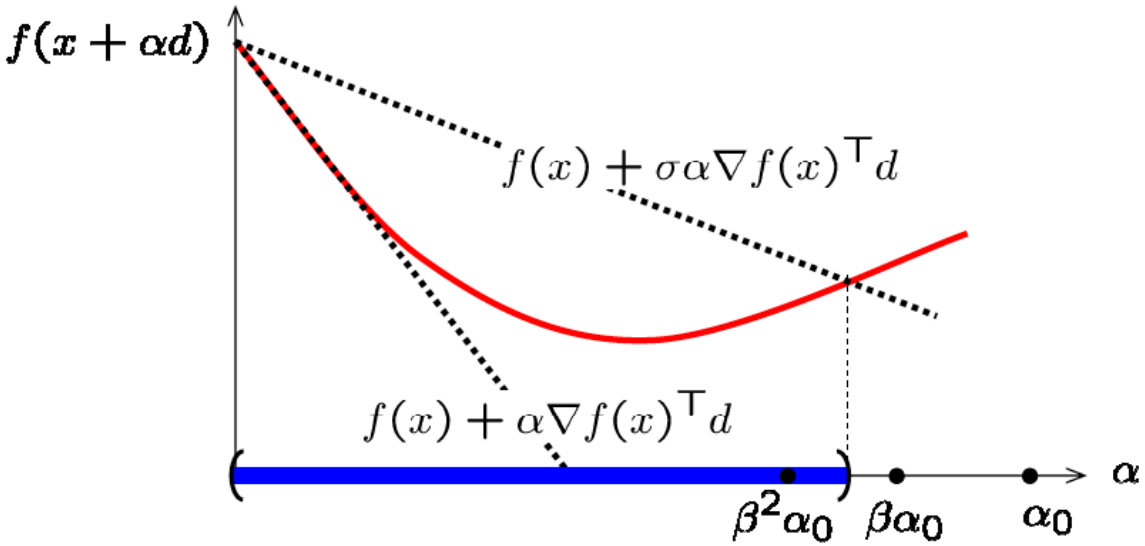
- Er som gradient descent, men skridtstørrelse vælges analytisk dvs. den optimale

$$\lambda_k = \underset{\lambda}{\operatorname{argmin}} f(\mathbf{x}_k - \lambda \nabla f(\mathbf{x}_k))$$

- Dette er virkelig den *ægte* stejleste nedstigningsretning da den sørger for at vi kigger, og sammenligner, i den samme *afstand* udaf hver mulig retning, uanset hældning
- Men som altid i datalogi, mere præcision koster beregningstid

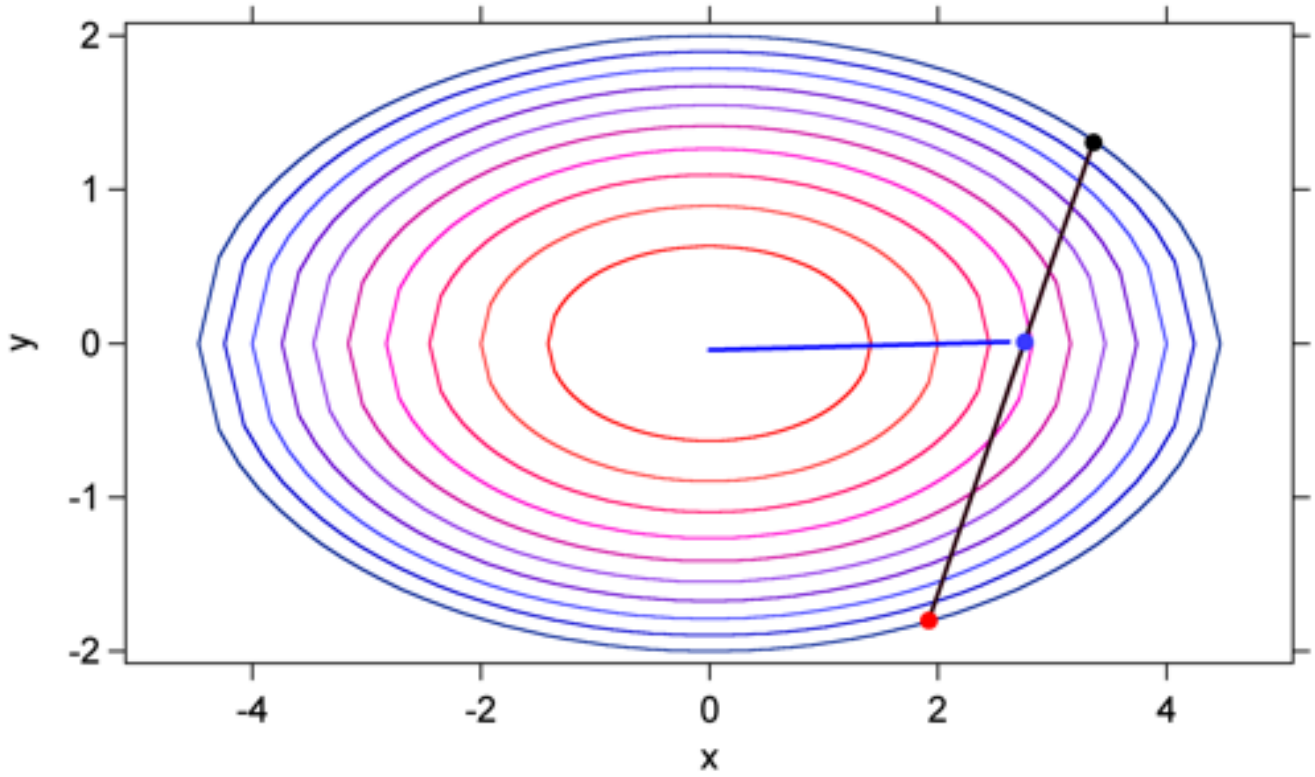
Steepest descent

Stejleste nedstigningsretning kan findes ved *line search*:
Start med en meget stor skridtstørrelse $\lambda_k = \lambda_0$ og gør den gradvist mindre ved at multiplicere med en værdi $\beta \in (0, 1)$ intil funktionen ikke længere er aftagende



Steepest descent

Line search illustreret for konturplot til 2-dimensionel funktion



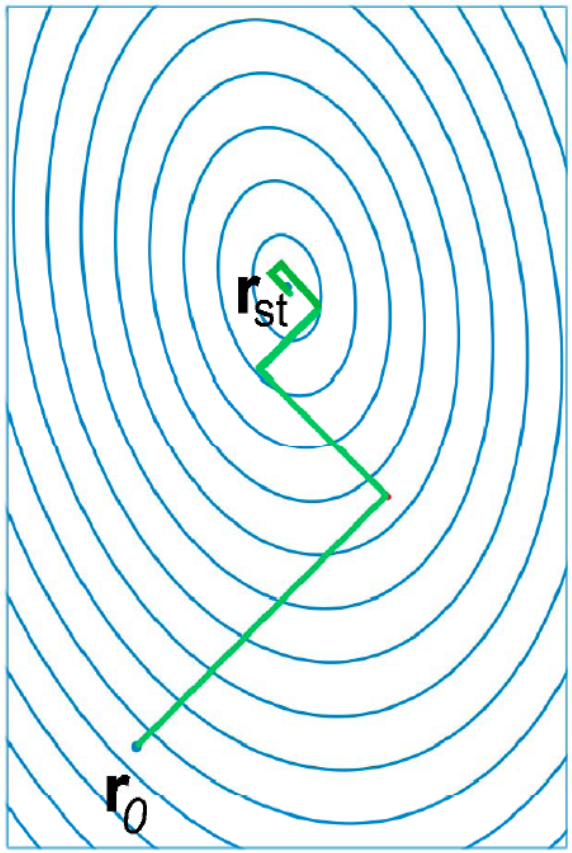
Newton algorithm

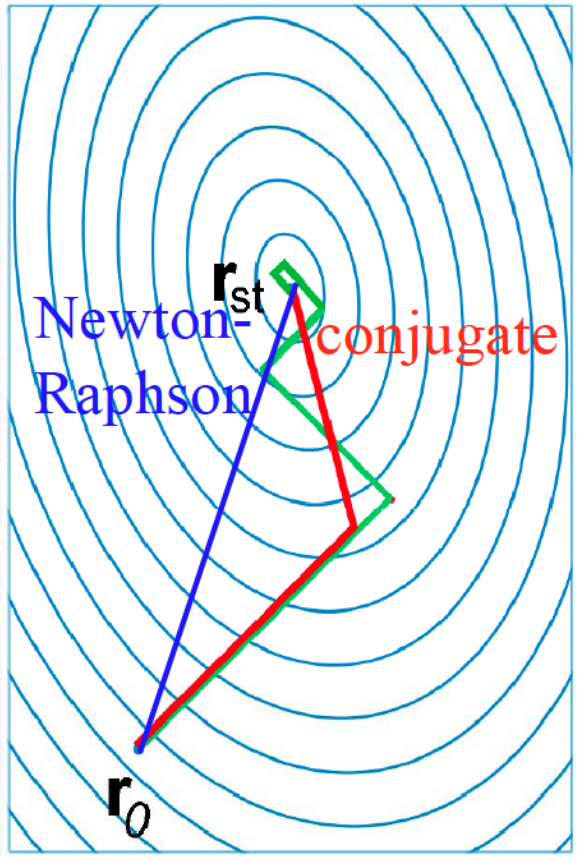
- Udvidelsen af 1-D metoden til det multidimensionelle: Givet $f(\mathbf{x})$, tilnærm f med en andensordens Taylor serie ved $\mathbf{x} = \mathbf{x}_k$:

$$f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}\nabla^2 f(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k)^2$$

- Algoritme som steepest descent, hvor nedstigningsretningen sættes til 'Newton retningen' $\mathbf{d}_k = -\left(\nabla^2 f(\mathbf{x}_k)\right)^{-1} \nabla f(\mathbf{x}_k)$
- Hessian matricen er generel dyr at beregne og dens inverse ikke ligetil, men konvergerer meget hurtigere end steepest descent

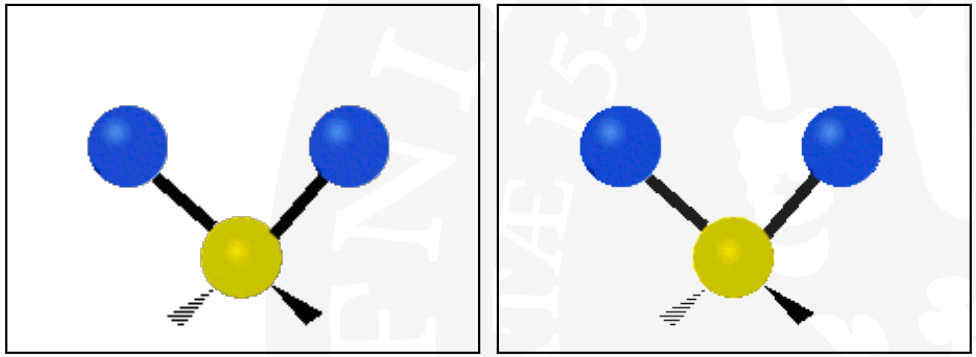
Newton algorithm vs. steepest descent





Eksempel: Molekulær dynamik

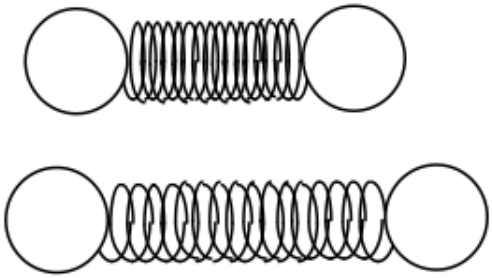
Lad os gå tilbage til den 1-dimensionelle verden for en kort stund og studere et eksempel på et optimeringsproblem:

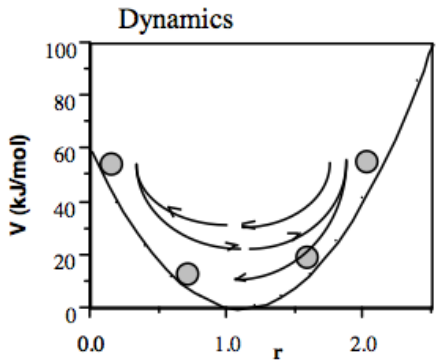


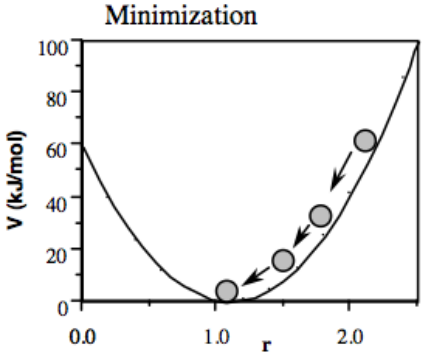
Diatomiske molekyler:

- Indeholder to atomer i et 'vibrerende' samspil - påvirker konstant hinanden med energi
- Hvad vil molekylet? Minimere sit energiforbrug!

Eksempel: Molekulær dynamik







- Molekylet har en *potentialenergifunktion* som summerer al atomar energipåvirkning
- Laveste energikonformation af molekylen findes ved at finde en position for alle atomer der minimerer den samlede energiladning - dvs. *minimere potentialenergifunktionen*
- Molekylet gør det helt af sig selv, vi vil bare simulere det ;)

Eksempel: Molekulær dynamik

I vores simple diatomiske molekyleeksempel beskæftiger potentialfunktionen sig kun om een ting: Længden af de såkaldte 'bonds' for atomerne samt den mindst energiforbrugende referencelængde:

- Bond stretching interaction: Energien af et bond ændres med dens længde
- Energien af en bond er lavest ved en bestemt referencelængde
- så vi minimerer afstanden mellem bond length og referencelængden

Eksempel: Molekulær dynamik

Dette kan vi skrive op i pæne formler

- r: nuværende bond length
- r_{eq}: equilibrium (reference) bond length (længde ved minimum energi)
- k: kraft (force) konstant for bond'et (hvor kraftigt hives atomet tilbage fra højenergitilstand)

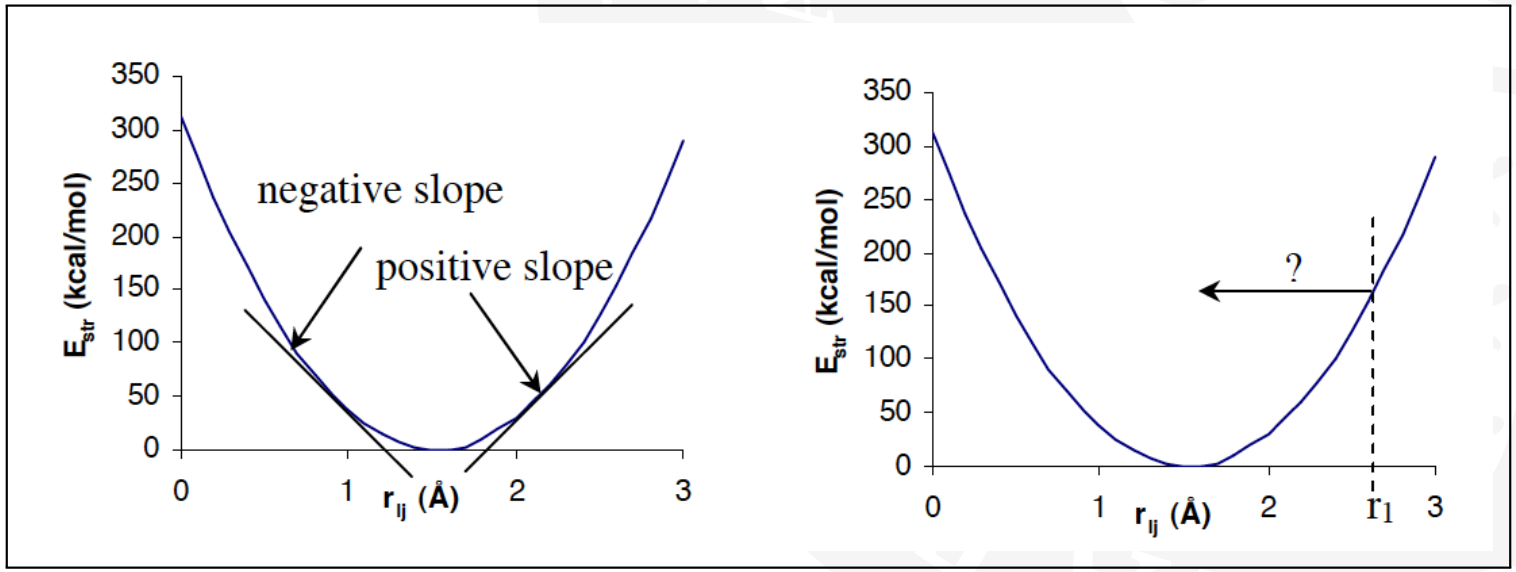
Så har vi

- Energifunktion $E = \frac{1}{2}k(r - r_{eq})^2$
- Den afledte $\frac{dE}{dr} = k(r - r_{eq})$
- k er givet ved $\frac{d^2E}{dr^2}$ for $r = r_0$

Ideen er nu at bruge gradient descent til at minimere E v.h.a. $\frac{dE}{dr}$

Eksempel: Molekulær dynamik

- Givet en start-bond længde r_1 skal vi bruge den afledte af potentialenergifunktionen til at finde vej mod den optimale længde
- Bemærk: Det er ikke et svært problem med eet optimeringsterm som her, ideen er at der typisk vil være mange termer, der skal optimeres over mange atomer



Eksempel: Molekulær dynamik

Descent minimering i Python

```
k = 2743.0   # Harmonisk kraft konstant
r_eq = 1.1283  # Equilibrium afstand

r = 1.55 # Initiel afstand
tau = 0.0001 # Skridtstørrelse
iterations = 50
E = 0.5 * k * (r - r_eq)**2  # Energi
F = k * (r - r_eq)           # Kraft (afledte)
```

Eksempel: Molekulær dynamik

Descent minimering i Python

```
# Descent search
for i in range(1, iterations):
    r = r - tau*F           # skridt
    #E = 0.5 * k * (r - r_eq)**2
    F = k * (r - r_eq)
    if (abs(F) < 0.01):  # Konvergenskriterie
        break
```

Eksempel: Molekulær dynamik

Descent minimering i Python - output

```
def E(k, r_eq, r):
    return 0.5 * k * (r - r_eq)**2

# Descent search
for i in range(1, iterations):
   r = r - tau*F          # skridt
   F = k * (r - r_eq)
   step_r[i] = r
   if (abs(F) < 0.01):  # Konvergenskriterie
      break
```

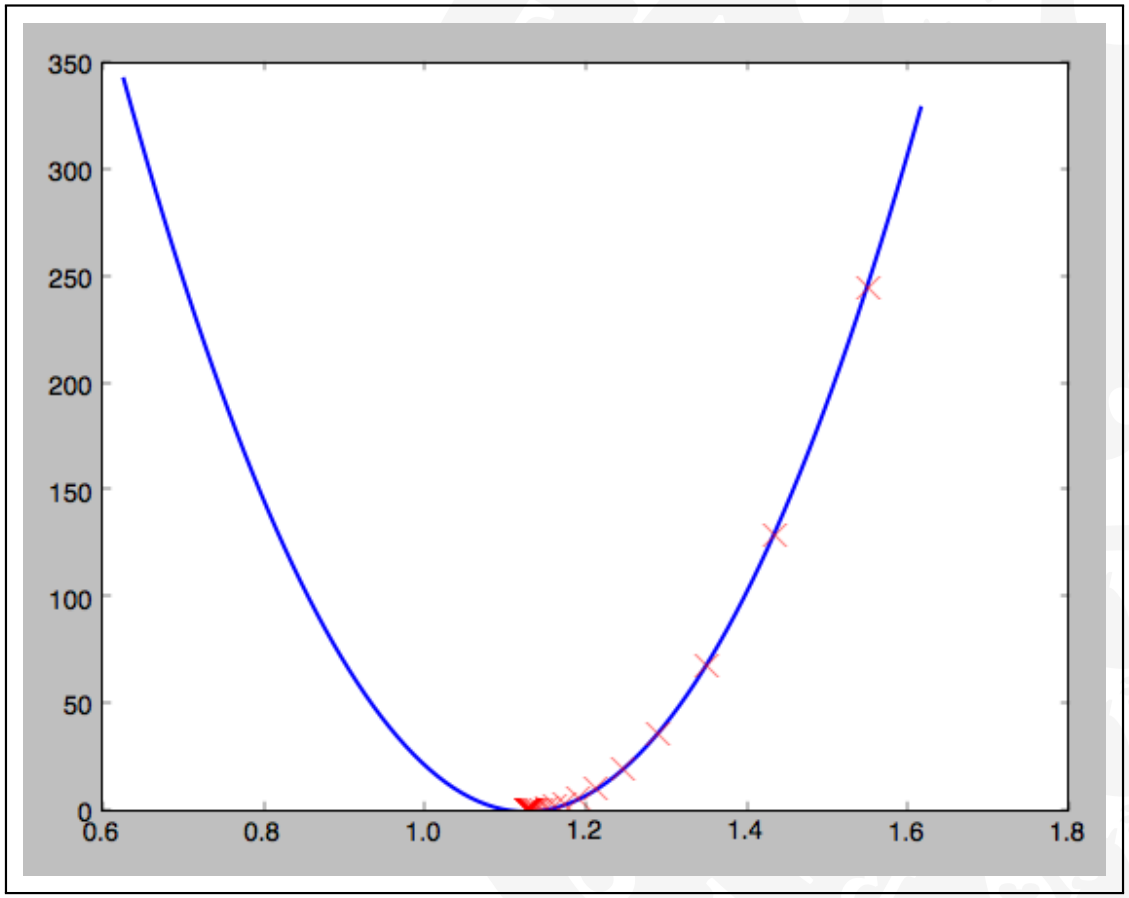
Eksempel: Molekulær dynamik

Descent minimering i Python - output

```
# Plot
fig = plt.figure()
plt.hold(True)
t = np.arange(r_eq-0.5, r_eq+0.5, 0.01)
plt.plot(t, E(t), '-')
plt.plot(step_r, E(step_r), 'rx')
plt.show()
```

Eksempel: Molekulær dynamik

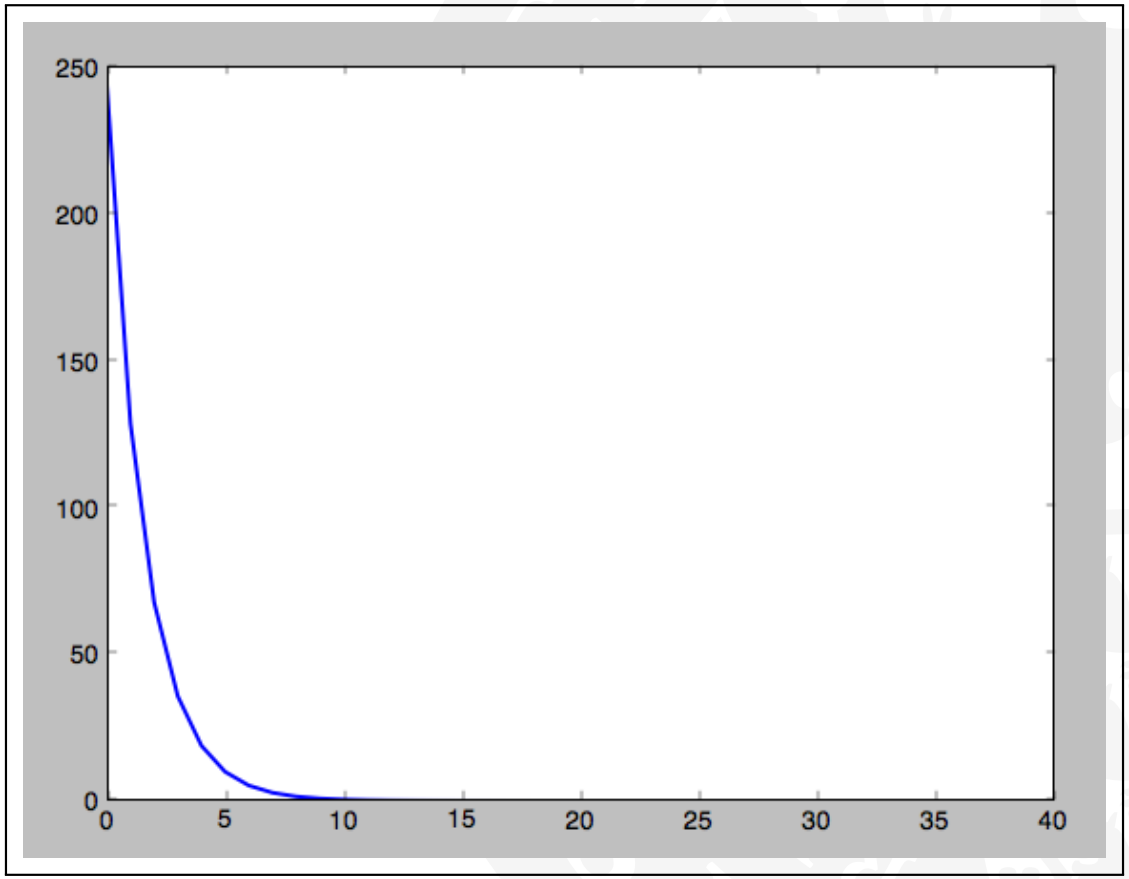
Descent minimering i Python




```
r_0 = 1.1283040950577616
```

Eksempel: Molekulær dynamik

Descent minimering i Python

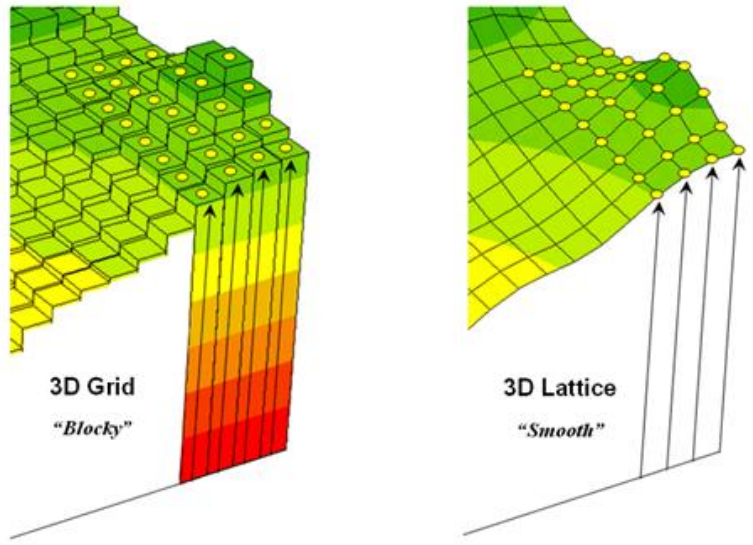


Diskretisering af gradient

Vi vil nu kigge på et lidt sværere problem, nemlig

- en funktion af 2 parametre
- beregning på *diskret* data

Formulering af problemer fra den virkelige verden vil typisk være baseret på målt, diskret data og metoderne til at navigere igennem det vil ligeledes være *diskrete*:



Diskretisering af gradient

Fra ugeseddelen:

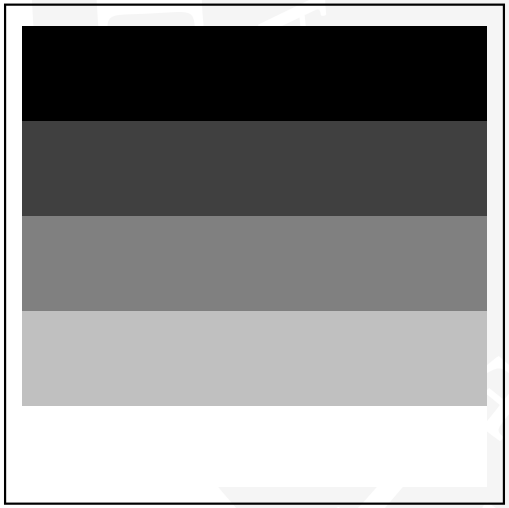
$$\nabla I_x(x_i, y_j) = \left\{ \begin{array}{ll} I(x_{i+1}, y_j) - I(x_i, y_j) & \text{if} \quad i < N \\ 0 & \text{if} \quad i = N \end{array} \right\} \tag{3}$$

$$\nabla I_y(x_i, y_j) = \left\{ \begin{array}{ll} I(x_i, y_{j+1}) - I(x_i, y_j) & \text{if} \quad j < N \\ 0 & \text{if} \quad j = N \end{array} \right\} \tag{4}$$

```
for i in range(N):
    for j in range(N):
        if j < (N-1):
            (imageListDx[i])[j] = (imageList[i])[j+1]
                                ...- (imageList[i])[j]
        else:
            (imageListDx[i])[j] = 0.0
```

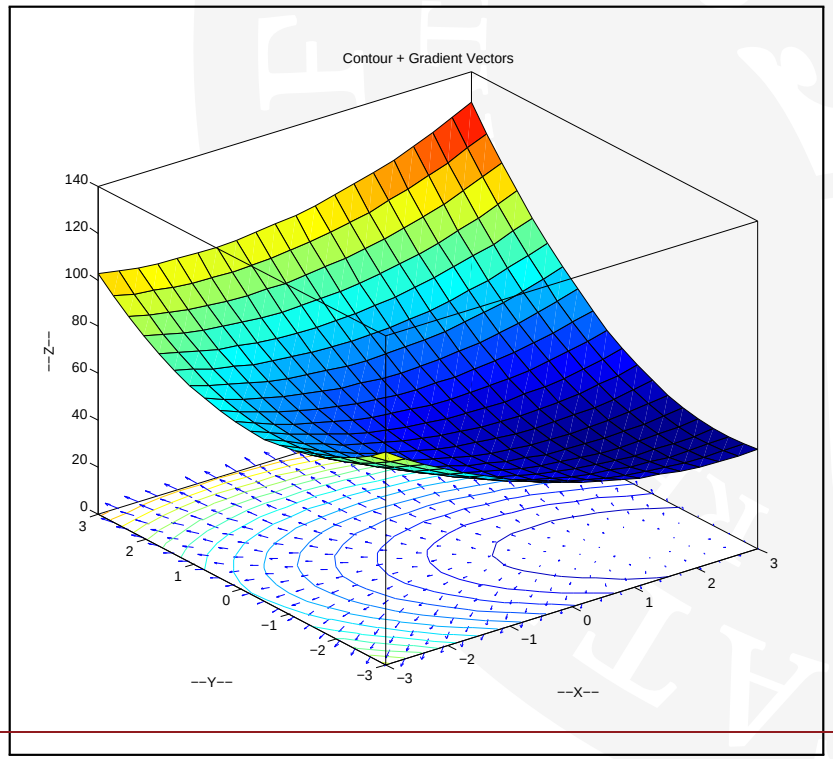
Diskretisering af gradient

$$\begin{bmatrix} -2 & -2 & -2 & -2 & -2 \\ -1 & -1 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$



Eksempel: Minimering af 2d funktion

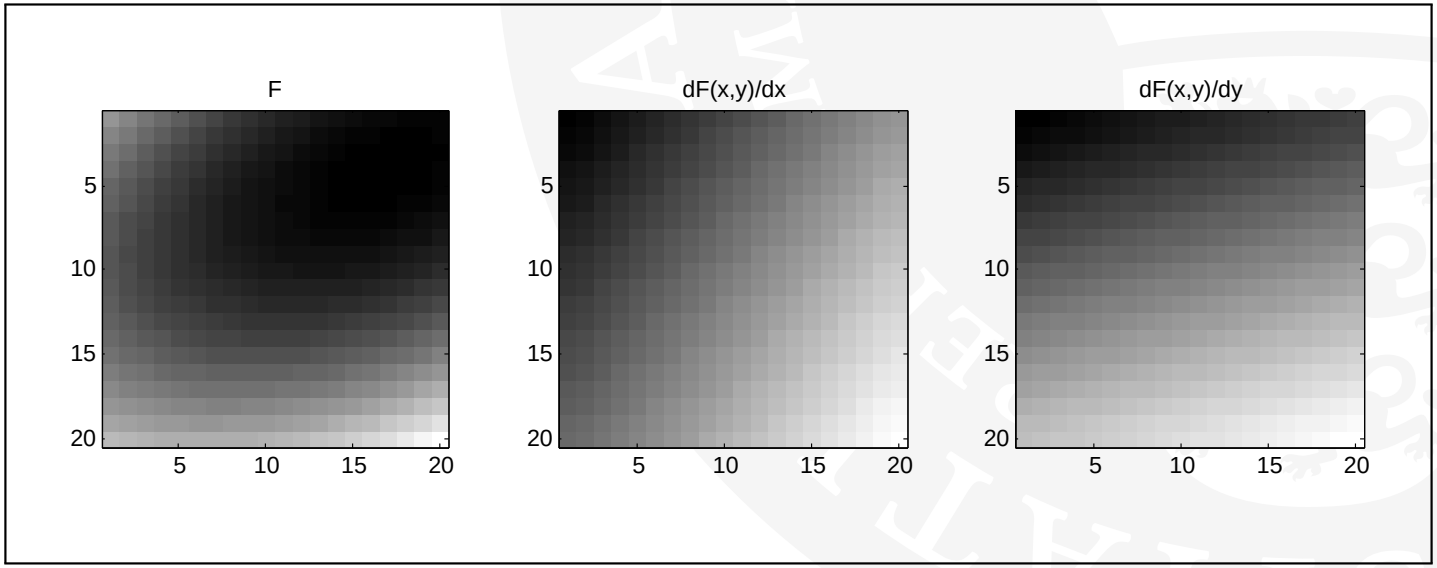
```
def fun(X, Y):
    return 0.5*(3*X**2 + 4*X*Y + 6*Y**2) - 2*X + 8*Y + 50
...
mesh_x = np.linspace(-3, 3, 20)
mesh_y = np.linspace(-3, 3, 20)
X, Y = np.meshgrid(mesh_x, mesh_y)
Z = fun(X, Y)
```



Eksempel: Minimering af 2d funktion

```
Zlist = Z.tolist()
ZlistArray = np.array(Zlist)
plt.imshow(ZlistArray, cmap="Greys_r")
plt.show()

ZListDx = gradientDx(Zlist)
...
```



Eksempel: Minimering af 2d funktion

```
iterations = 100
tau = 0.1

x0 = [0] * iterations # steps
y0 = [0] * iterations
x0[0] = 2 # start position
y0[0] = 2
x0mesh = [0] * iterations # step mesh index
y0mesh = [0] * iterations
idx_x = np.argmin(np.abs(mesh_x - x0[0]))
idx_y = np.argmin(np.abs(mesh_y - y0[0]))
x0mesh[0] = idx_x
y0mesh[0] = idx_y
```

Eksempel: Minimering af 2d funktion

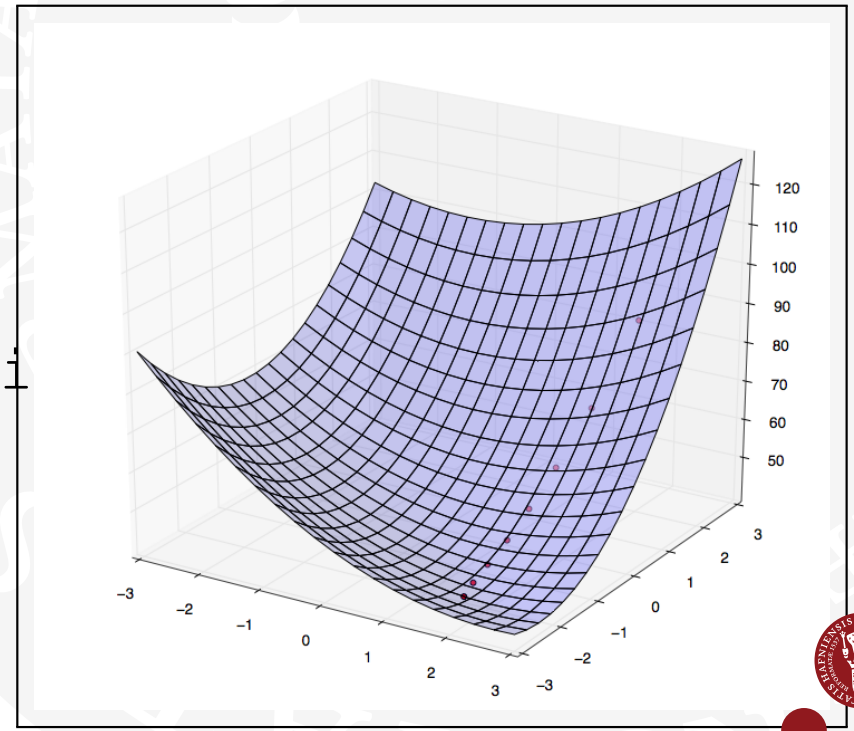
```
for it in range(1,iterations):
    idx_x = np.argmin(np.abs(mesh_x - x0[it-1]))
    idx_y = np.argmin(np.abs(mesh_y - y0[it-1]))
    x0mesh[it] = idx_x
    y0mesh[it] = idx_y
    x0[it] = x0[it-1] - tau*ZListDxArray[idx_x, idx_y]
    y0[it] = y0[it-1] - tau*ZListDyArray[idx_x, idx_y]

    if fun(x0[it], y0[it]) > fun(x0[it-1], y0[it-1]):
        break
```

Eksempel: Minimering af 2d funktion

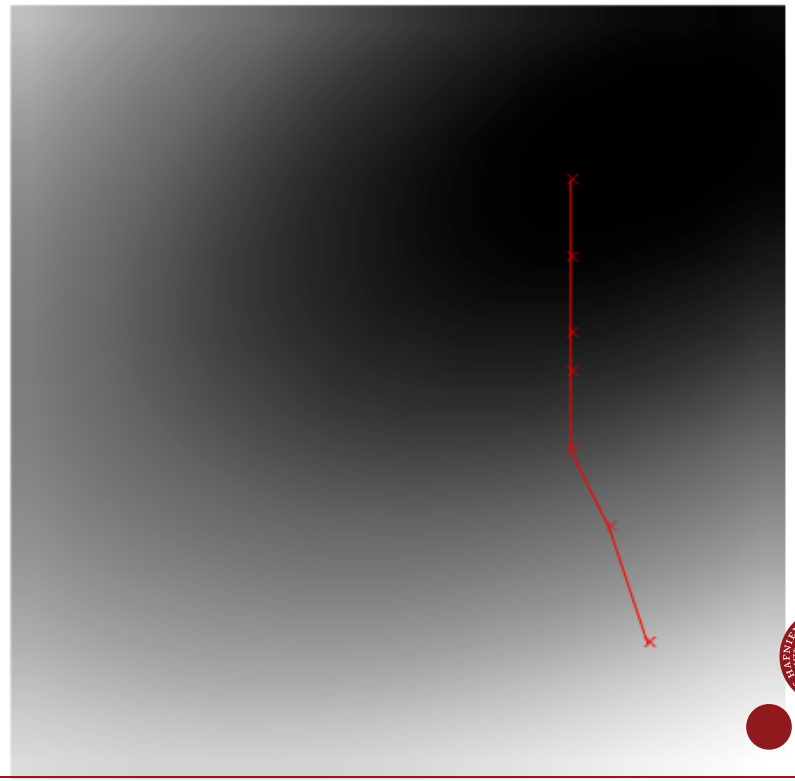
```
fig = plt.figure()
ax = fig.gca(projection='3d')
plt.hold(True)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, alpha
ax.scatter(x0[0:it], y0[0:it], z0, c='r', marker='o')
plt.show()

z0 = [0] * (it)
for it2 in range(0,it):
    z0[it2] = fun(x0[it2], y0[i
```



Eksempel: Minimering af 2d funktion

```
fig = plt.figure()
plt.hold(True)
plt.imshow(ZlistArray, cmap="Greys_r")
plt.plot(x0mesh[0:it], y0mesh[0:it],'rx')
plt.plot(x0mesh[0:it], y0mesh[0:it],'r-')
plt.show()
```



Ugeopgave: Fjernelse af støj fra billede

TAVLE :)