Programmering og Modellering (PoM)
Ugeseddel 8 — Uge 44 — Deadline 31/10

Kim Steenstrup Pedersen, Katrine Hommelhoff Jensen, Knud Henriksen,
Mossa Merhi og Hans Jacob T. Stephensen

October 23, 2014

1 Plan for ugen

I denne uge tager vi fat på *optimering*, hvilket kort sagt svarer til at minimere eller maksimere en funktion, dvs. finde punkter hvor funktionen antager et lokalt eller globalt minima eller maksima. For en endimensionel funktion $f(x)$ ved vi at et ekstremum kan findes blandt rødderne som fremkommer ved at løse $f'(x) = 0$. For flerdimensionelle funktioner $f(x_1, x_2, \ldots, x_n)$ er dette mere kompliceret og man anvender istedet *optimeringsalgoritmer* til at finde et minima/maksima. De mest generelt anvendelige optimeringsalgoritmer er *iterative metoder*, som trinvist søger gennem løsningsrummet, hvor der for hvert trin bliver taget en beslutning om retning og afstand, baseret på bl.a. egenskaber ved det nuværende ståsted. Billedligt talt kan løsningsrummet ses som et bjerglandskab og algoritmen en bjergvandrer på udkig efter den første og bedste dal (lokalt minimum) eller den aller dybeste af alle (globalt minimum), hvis næste trin f.eks. afgøres af landskabets hældning. De mange iterative optimeringsalgoritmer kan typisk kategoriseres som enten deterministiske (lokale) eller stokastiske (globale), hvor de deterministiske udelukkende er baseret på funktionens første- og andenordens afledte, mens de stokastiske indkorporerer tilfældigheder i valget af rute genem løsningsrummet. På turen gennem bjerglandskabet vil de deterministiske metoder for hvert trin vælge den retning, som set lokalt omkring det nuværende ståsted går mest ned ad bakke. Hvis vi allerede er tæt på den ønskede dal er dette den hurtigste måde at komme nedad, men vi risikerer også at sidde fast i et lille hul oppe i bjergene. Risikoen for at sidde fast i sådan et lokalt minimum er mindre ved de stokastiske metoder, hvor retning og skridtstørrelse for næste trin afgøres udfra informationer om landskabet længere væk fra nuværende ståsted, uden dog at have undersøgt *alle* muligheder i den afstand, men baseret på et par tilfældige retninger. Dette svarer til at kunne se udover det lille hul på bjerget, men ikke nødvendigvis tage den helt optimale rute ud af hullet. Dertil tager det potentielt længere tid at nå ned i dalen, da bjergvandreren har en tendens til at zig-zagge på sin vej.

Gradient descent

Centralt for ugeopgaven og øvelserne i denne uge er metoden *gradient descent*. Gradient descent er en deterministisk optimeringsalgoritme der kan bruges til at finde minima af en funktion f, dvs. at løse $\underset{x}{\text{argmin}} f(\mathbf{x})$, for $\mathbf{x} \in \mathbb{R}^n$. Bemærk at vi med de deterministiske metoder kun kan garantere et globalt minima hvis f er konveks; ellers et lokalt. Gradient descent opdaterer iterativt et bud på det $\mathbf{x}$ der minimerer f:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \tau_k \nabla f(\mathbf{x}_k) \tag{1}$$

hvor $\tau_k > 0$ er en skalarværdi for skridtstørrelsen, $\mathbf{x}_k$ er det nuværende punkt (ståsted), $\mathbf{x}_{k+1}$ det næste punkt og $\nabla f(\mathbf{x}_k)$ gradienten i nuværende punkt, givet ved

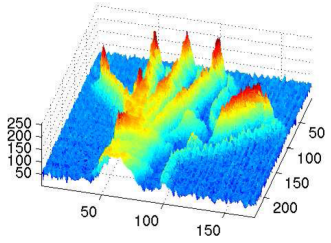
$$\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}(\mathbf{x}), \frac{\partial f}{\partial x_2}(\mathbf{x}), \ldots, \frac{\partial f}{\partial x_n}(\mathbf{x}) \right)$$

som udgør en vektor der peger i retningen af den største stigning. Ved at tage et skridt i retningen af den negative gradient, foretager vi således en nedstigning. Som standard sættes skridtstørrelsen

manuelt, men kan også estimeres i hver iteration. Beregnes den bedst mulige skridtstørrelse, kaldes metoden for *steepest descent*.

Diskrete todimensioneller funktioner i Python

Data indsamlet fra den virkelige verden vil typisk være repræsenteret i en *diskret* form, dvs. indsamlet i diskrete punkter. Selvom man eventuelt kunne udlede en korresponderende, kontinuert funktion, viser det sig ofte mere effektivt at arbejde direkte på den diskrete data og anvende *numeriske* metoder til at beregne på den. Et godt eksempel på diskret data er billeder. Et 2d billede er indsamlet i diskrete dataenheder kaldet *pixels* og kan repræsenteres som en diskret billedfunktion $I(x, y)$, som tager en position i billedet (x, y) og returnerer en farveintensitet for den tilsvarende pixel. Farveintensiteten har typisk en værdi mellem 0 og 255, hvor høje tal svarer til lyse farver. Dette er illustreret nedenfor, hvor et gråtonebillede (venstre) er plottet som en diskret funktion (højre), med en kunstig farveskala for at tydeliggøre funktionens struktur:



I forbindelse med optimeringsalgoritmerne skal vi arbejde med diskrete, todimensionelle funktioner, heriblandt billeder, og skrive numeriske metoder til at beregne på disse funktioner. I Python kan et billede f.eks. repræsenteres ved en liste af lister, dvs.

$$\left[\begin{bmatrix} (x_0, y_0) \\ (x_1, y_0) \\ \vdots \\ (x_n, y_0) \end{bmatrix}, \begin{bmatrix} (x_0, y_1) \\ (x_1, y_1) \\ \vdots \\ (x_n, y_1) \end{bmatrix}, \ldots \begin{bmatrix} (x_0, y_m) \\ (x_1, y_m \\ \vdots \\ (x_n, y_m) \end{bmatrix}\right] \tag{2}$$

for et billede med n rækker og m søjler. Erklæres i Python listen af lister `imageList` kan man således tilgå intensitetsværdien i punktet $(x, y) = (3, 5)$ ved at kalde `(mImageList[3])[5]`. Billedet kan plottes ved at kalde

```
import matplotlib.pyplot as plt
plt.imshow(imageList)
plt.show()
```

For gråtoner kaldes

```
plt.imshow(imageList, cmap="Greys_r")
plt.show()
```

Til tirsdag:

Læs: Numerical Optimization, kap. 1: Introduction (noter)
Til forelæsningen gennemgås:

- Introduktion til optimering
- Multidimensionelle funktioner og deres afledte
- Konveksitet, lokale og globale ektremum
- Deterministiske (lokale) optimeringsalgoritmer

Til torsdag:

Læs: Guttag kap. 13 (Random walks) og 14 (Monte Carlo simulation).
Til forelæsningen gennemgås:

- Deterministiske (lokale) optimeringsalgoritmer (fortsat)
- Stokastiske (globale) optimeringsalgoritmer

1.1 Gruppeopgave

Den 31/10 senest klokken 15:00 skal besvarelse af følgende opgave afleveres elektronisk via Absalon. Opgaven skal besvares i grupper bestående af 1 til 3 personer og skal godkendes, for at gruppedeltagerne kan kvalificere sig til den afsluttende tag-hjem eksamen. Opgavebesvarelsen skal oploades via kursushjemmesiden på Absalon (find underpunktet `ugeseddel8` under punktet `Ugesedler og opgaver`). Kildekodefiler ("script"-filer) skal afleveres som "ren tekst", sådan som den dannes af `emacs`, `gedit`, `Notepad`, `TextEdit` eller hvilket redigeringsprogram man nu bruger (*ikke* PDF eller HTML eller RTF eller MS Word-format). Filen skal navngives *efternavn1.efternavn2.efternavn3*.44.py, mens andre filer skal afleveres som en PDF-fil med navnet *efternavn1.efternavn2.efternavn3*.44.pdf.

8g1 I denne opgave skal I arbejde med den deterministiske optimeringsalgoritme *gradient descent* samt et klassisk problem fra billedbehandling; at fjerne støj i et billede.

Gradient og divergens i billeder

Lad $I(x, y)$ være en diskret 2d billedfunktion som beskrevet tidligere. For at simplificere antager vi at $I(x, y)$ har samme antal pixels N i bredden og højden. Vi vil nu se på hvordan lokale differentialoperatorer som *gradienten* og *divergensen* kan bruges til at beskrive bestemte egenskaber ved billedet og hvordan operatorerne beskrives diskret. Hvor en kontinuert funktion $f \in \mathbb{R}^2$ har gradienten $\nabla f(x_i, y_j) = \left(\frac{\partial f}{\partial x}(x_i, y_j), \frac{\partial f}{\partial y}(x_i, y_j)\right) \in \mathbb{R}^2$ i punktet (x_i, y_j), kan den korresponderende diskrete gradientoperator defineres for billedet $I(x, y)$ som

$$\nabla I(x_i, y_j) = (\nabla I_x(x_i, y_j), \nabla I_y(x_i, y_j)) \in \mathbb{R}^2 \tag{3}$$

hvor

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

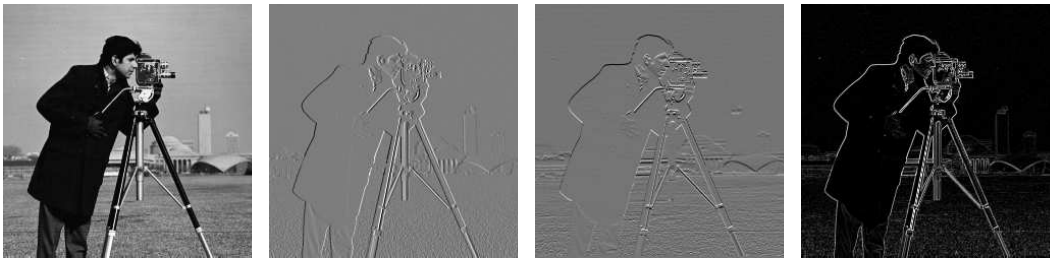
$$\nabla I_y(x_i, y_j) = \left\{ \begin{array}{ll} I(x_i, y_{j+1}) - I(x_i, y_j) & \text{if } j < N \\ 0 & \text{if } j = N \end{array} \right\} \tag{5}$$

for $i, j = 1, \ldots, N$. $\nabla I(x_i, y_j)$ består således af to gradientbilleder $\nabla I_x(x_i, y_j)$ og $\nabla I_y(x_i, y_j)$. I Python kunne man f.eks. skrive $\nabla I_x(x_i, y_j)$ ved at løbe gennem alle billedelementerne:

```
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
```

hvor `imageList` er $I(x, y)$ repræsenteret som en liste af lister og `imageListDx` tilsvarende for $\nabla I_x(x_i, y_j)$.

Betragt billederne på Fig. 1. På Fig. 1i ses det originale billede og på Fig. 1ii og 1iii de partielt afledte $\nabla I_x(x_i, y_j)$ og $\nabla I_y(x_i, y_j)$, henholdsvist. Bemærk hvordan gradienten fanger *kanterne* i billedstrukturen. Tilstedeværelsen af kanter tydeliggøres yderligere i Fig. 1iv, som



(i) Billede I (ii) $\nabla I_x(x_i, y_j)$ (iii) $\nabla I_y(x_i, y_j)$ (iv) Normen

Figure 1: Billede, dets afledte og normen.

illustrerer normen (længden) af gradienten, givet ved operatoren

$$||\nabla I(x_i, y_j)|| = \sqrt{\nabla I_x(x_i, y_j)^2 + \nabla I_y(x_i, y_j)^2} \tag{6}$$

der returnerer et billede hvor en lys farve indikerer en høj intensitetsværdi, reflekteret af en lang gradient og derved en stejl kant i det originale billede. På gradientbillederne (eller andre billeder hvis værdier beskriver retninger) kan man yderligere tage *divergensen*, en operator $\text{div} = -\nabla^*$ som i sin diskrete form er givet ved

$$\begin{aligned} \text{div}(v(x_i, y_j), w(x_i, y_j)) = &\left\{ \begin{array}{rll} v(x_i, y_j) - v(x_{i-1}, y_j) & \text{if} & 1 < i < N \\ v(x_i, y_j) & \text{if} & i = 1 \\ -v(x_{i-1}, y_j) & \text{if} & i = N \end{array} \right\} \\ + &\left\{ \begin{array}{rll} w(x_i, y_j) - w(x_i, y_{j-1}) & \text{if} & 1 < j < N \\ w(x_i, y_j) & \text{if} & j = 1 \\ -w(x_i, y_{j-1}) & \text{if} & j = N \end{array} \right\} \end{aligned} \tag{7}$$

og som fortæller i hvor høj grad retningerne omkring $(v(x_i, y_j), w(x_i, y_j))$ konvergerer (peger ind mod punktet) eller divergerer (peger væk fra punktet), hvor $v(x_i, y_j)$ og $w(x_i, y_j)$ kan erstattes med gradientbillederne. Gradient- og divergensoperatorerne er meget nyttige for detektionen af f.eks. kanter, højderygge og ekstrema i billeder.

Opgave: Fjernelse af støj fra billeder

For at løse opgaven skal du bruge følgende data (læg filerne i samme folder som dit Python program):

- Filer med billeddata `Cameraman.csv` og `CameramanNoisy.csv`
- Python filen `csvImageRead.py`

(a) Indlæs billedet `cameraman.csv` v.h.a funktionen `csvImageRead.py`, der returnerer pixelværdierne i en liste af lister:

```
from csvImageRead import csvImageRead
imageList = csvImageRead("Cameraman.csv")
```

Det skal derudover afprøves om pixelværdierne er mellem 0 og 255, samt at bredden af billedet er lig højden. Plot billedet (hint: som beskrevet i introduktionen).

(b) Skriv en funktion `gradient(V)` der tager et billede (en liste af lister) og returnerer gradienten i form af de to partielt afledte `dVx` og `dVy`, også repræsenteret som billeder, som beskrevet i (4) og (5) - hent inspiration fra kodeeksemplet lige nedenunder formel (5). Beregn gradienten af det indlæste billede (a), plot de resulterende billeder og tjek efter at de ser ud som i Fig. 1ii og 1iii.

(c) Skriv en funktion `gradNorm(V1, V2)` der tager to gradientbilleder og returnerer normbilledet som beskrevet i (6). Beregn normen af gradienten af det indlæste billede (se punkt (a)-(b)) og plot resultatet. Tjek at det ser ud som på Fig. 1iv.

(d) Skriv en funktion `divergence(V1, V2)` der tager to billeder og returnerer divergensen i et billede `divV`, som beskrevet i (7). Beregn divergensen af gradienten af det indlæste billede (se punkt (a)-(b)) og plot resultatet. Beskriv hvilke egenskaber i billedet, div∇ fanger.

(e) Indlæs nu billedet `CameramanNoisy.csv` som i (a):

```
imageNoisyList = csvImageRead("CameramanNoisy.csv")
```

Dette billede er, som illustreret nedenfor, billedet fra før pertuberet med støj:

Opgaven er nu at fjerne støjen fra billedet ved hjælp af (a)-(d) samt gradient descent metoden. Lad os sige at $\mathbf{y}$ er det diskrete, støjfyldte billede af størrelsen $N \times N$. Man kan da reducere støjen ved at minimere funktionen

$$\min_x f(\mathbf{x}) = \frac{1}{2}||\mathbf{y} - \mathbf{x}||^2 + \lambda J(\mathbf{x}) \tag{8}$$

hvor $\mathbf{x}$ er det støjreducerede billede og $J(\mathbf{x}) \in \mathbb{R}$ et udtryk for $\mathbf{x}$'s *glathed*, baseret på gradienten. Vi finder altså $\mathbf{x}$ som det billede med den minimale afstand til $\mathbf{y}$, som samtidig har glatheden fordret af det regulerende udtryk $\lambda J(\mathbf{x})$, hvor $\lambda > 0$ bestemmer udtrykkets indflydelse. Optimeringen af billedet $\mathbf{x}$ (8) kan letteregøres ved istedet at optimere

$$\min_W \frac{1}{2}||\lambda\mathbf{y} - \text{div}(W)||^2 \tag{9}$$

hvor W er en særlig type billede (kaldet vektorfelt), der ligesom gradienten indeholder information om bevægelsesretninger, og relaterer til $\mathbf{x}$ som

$$\mathbf{x} = \mathbf{y} - \frac{1}{\lambda}\text{div}(W). \tag{10}$$

Kort forklaret er div(W) et led som fjerner 'uglatheder' fra $\mathbf{y}$, størrelsen af hvilke reguleres af $\lambda > 0$. I skal nu finde $\mathbf{x}$ ved at minimere (9) via gradient descent som beskrevet tidligere. Dertil har I brug for gradienten til (9), som er givet ved

$$\nabla\left(\lambda\mathbf{y} - \text{div}(W)\right) = \nabla(\lambda\mathbf{y}) + \nabla(-\text{div}(W)) \tag{11}$$

Implementer algoritmen med følgende trin:

1) Indlæs det støjfyldte billede `CameramanNoisy.csv` (se punkt (e)) i variablen `y` og gem bredden i `N`
2) Sæt skridtstørrelse `tau` $= 0.248$ og regulering `lambda` $= 0.08$
3) Kreer to $N \times N$ billeder `w1` og `w2` som kun indeholder 0'er og beregn divergensen `divW` af dem (d). `w1` og `w2` repræsenterer W.
4) Beregn billedet $\lambda\mathbf{y}$ (multiplikationen af λ med alle billedværdierne i $\mathbf{y}$) og gem det i `ylambda`

5) I 200 iterationer, gør:
   5.1) Beregn gradientbillederne `dWx, dWy` for $\nabla(\lambda \mathbf{y} - \mathrm{div}(W))$ (11), ved at kalde `gradient(V)` (b) på h.h.v. `divW` og `ylambda` (det er vigtigt at beregne disse gradienter separat!)
   5.2) Beregn normen `dWnorm` af gradienten `dWx, dWy` v.h.a `gradNorm(V1, V2)` (c)
   5.3) Foretag descent skridt (1) på `w1` og `w2` v.h.a. `dWx`, `dWy` og `tau`. Normaliser `w1` og `w2` ved at dividere værdierne med (1 + `dWnorm*tau`)
   5.4) Opdater `divW` med `w1` og `w2` (d)
6) Få det resulterende billede `x` v.h.a. (10) og plot det

(f) Algoritmen (e) skal udvides med et trin (5.5) der beregner og gemmer værdien af udtrykket i (9) for hver iteration, i en liste. Hint: beregn normen i udtrykket som $||v(x,y) - w(x,y)||^2 = \sum_i^N \sum_j^N (v(x_i, y_j) - w(x_i, y_j))^2$. Plot resultatet som en graf. Opfører det sig som forventet?

(g) Hvad er effekten ved at variere størrelsen på λ i algoritmen (e)?

1.2 Tirsdagsøvelser

Besvarelser af disse opgaver skal ikke afleveres, men opgaverne forventes løst inden torsdag.

En kontinuert funktion $f(x,y) = x^2 + y^2$ kan diskretiseres og plottes i Python som f.eks. (hvad der sker i koden bliver forklaret senere på kurset):

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def fun(X, Y):
    return X**2 + Y**2

fig = plt.figure()
ax = fig.gca(projection='3d')
plt.hold(True)
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
Z = fun(X, Y)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, alpha=0.2)
plt.show()
```

og de diskrete billedværdier kan herefter fås som en liste af lister ved at kalde `Z.tolist()`.

8ti1 Diskretiser og plot funktionen $f(x,y) = x^2 + y^2$ som vist ovenfor. Beregn gradientbillederne ved at skrive funktionen `gradient(V)` som det også bedes om i ugeopgave (b), og kalde den med `Z.tolist()` (hint: læs opgaveteksten først). Plot gradientbillederne. Beskriv relationen mellem værdierne i gradientbillederne og f, herunder position af minimum.

8ti2 Implementer den basale gradient descent mimimering af f som i (1), ved at bruge `gradient(V)` til at beregne ∇f. Vælg en startposition (x_0, y_0), fast skridtstørrelse τ samt antal iterationer. Gem positionen opnået i hvert skridt og plot dem ovenpå overfladen for f. Et eksempel på plotning af punkter er

```
x =[1, 3, 5, 2.3]
y =[0.5, 2, 1, 4]
z =[4, 6, 20, 3]
ax.scatter(x, y, z, c='r', marker='o')
```

Her skal `x`, `y` og `z` skiftes ud med skridtkoordinaterne og kaldet til `ax.scatter(x, y, z, c='r', marker='o')` indsættest lige før `plt.show()`. Argumenter for om det globale minima blev fundet (hint: brug den kontinuerte afledte af f). Beskriv hvordan valget af skridtstørrelse, antal iterationer og startposition hver for sig har indflydelse på om minima kan nås.

8ti3 Indfør et passende konvergenskriterie istedet for et fast antal iterationer (f.eks. 1) kriteriet for den afledte i et punkt, 2) om skridtene konvergerer til det samme punkt), og kør minimeringen igen. Gem energien (værdien af $f(x^*, y^*)$ for det nuværende minima (x^*, y^*)) for hver iteration og plot energierne efter konvergering som en graf.

8ti4 Kør nu minimeringen på den diskretiserede $f(x, y) = \frac{x-y}{x^2+y^2+2}$. Plot funktionen med skridt og det fundne minimum. Hvordan har startpositionen en indflydelse på om det globale minima kan findes?

1.3 Torsdagsøvelser

Besvarelser af disse opgaver skal ikke afleveres, men opgaverne forventes løst inden tirsdag i efterfølgende uge.Vi fortsætter med øvelserne fra tirsdag (start med dem hvis du ikke blev færdig).

8to1 Gradient descent algoritmen fra sidste opgave kan udvides til at beregne en passende skridtstørrelse for hver iteration. *Steepest descent* er en sådan metode, hvor skridtstørrelsen τ_k i hvert skridt findes som minima af

$$\varphi(\tau_k) = f(\mathbf{x}_k - \tau_k \nabla f(\mathbf{x}_k)) \tag{12}$$

Skriv en forsimplet version af $\varphi(\tau_k)$ hvor τ_k findes som den ud af 10 forskellige værdier, der giver den stejleste nedstigning. Kør algoritmen på $f(x, y) = x^2 + y^2$ og plot energi-grafen. Hvad sker der for konvergensraten?

8to2 Gradient-baserede og andre *lokale* minimeringsalgoritmer er sensitive overfor lokale minima. Forklar hvorfor?

8to3 Diskretiser og plot Rosenbrock funktionen $f(x, y) = (1-x)^2 + 100(y-x^2)^2$ på samme måde som i de foregående øvelser. Det har sit minima i $(x, y) = (1, 1)$. Kør gradient descent og steepest descent minimeringsalgoritmerne og plot skridt samt konvergering. Hvorfor er det svært at finde minima for denne funktion for de gradient-baserede metoder (hint: kig på gradienterne)?