Prosjekt 3 - Introduksjon til Vitenskapelige Beregninger

Studentnr: 755110, 759144 og 753717

April 2016

1 Oppgave 1

Røntgenstråler emittert fra en wolfram-anode har en karakteristisk energi $E_k \approx 60\,\text{keV}$, noe som tilsvarer en bølgelengde

$$\lambda = \frac{hc}{E_k} \approx 0.021\,\text{nm}, \tag{1}$$

der h er Plancks konstant, og c er lysets hastighet i vakum.

2 Oppgave 2

En tynn, monokromatisk røntgenstråle med intensitet I_0 sendes inn mot overkroppen, slik at fotonene passerer 10 cm mykt vev, deretter 1 cm bein, og til slutt 9 cm mykt vev. Dette tilsvarer en lineær attenuasjonskoeffisientfordeling på

$$\mu(t) = \begin{cases} \mu_v = 20\,\text{m}^{-1} & t \in [0, 0.1] \cup [0.11, 0.2] \\ \mu_b = 600\,\text{m}^{-1} & t \in [0.1, 0.11] \end{cases}. \tag{2}$$

t er gitt i meter (m), og μ_v og μ_b er attenuasjonskoeffisientene til henholdsvis mykt vev og bein. Røntgenstrålen går inn i overkroppen ved $t_1 = 0\,\text{m}$ og ut ved $t_2 = 0.2\,\text{m}$. Projeksjonen p av overkroppen kan så beregnes ved

$$p = \ln \frac{I_0}{I} = \int_{t_1}^{t_2} \mu(t)\,\mathrm{d}t = 9.8, \tag{3}$$

der $\mu(t)$ er gitt ved ligning (2). Reduksjonen i strålens intensitet ved transmisjon kan så beregnes

$$\frac{I}{I_0} = e^{-p} \approx 5.5 \times 10^{-5}. \tag{4}$$

Siden attenuasjonskoeffisiententene er lineære, dvs. $\mathrm{d}\mu(t)/\,\mathrm{d}t = 0$, så vil rekkefølgen til plasseringen av det myke vevet og beinet ikke ha noen innvikrning på verdien til p og I så lenge *total* mengde med vev og bein forblir likt.

3 Oppgave 3

Et kvadratisk bein i midten av et kvadratisk legeme med mykt vev kan representeres av en kvadratisk matrise hvor hvert element har verdien

$$f(x_i, y_j) \equiv f_{ij} = \begin{cases} 1 & \text{(hvitt; bein)} \\ 0 & \text{(svart; mykt vev)} \end{cases}. \tag{5}$$

Med 3×3 piksler kan derfor objektet beskrives av en 3×3 matrise

$$M = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{6}$$

Ved standard matrisenotasjon har vi derfor sammenhengen $f_{ij} = M_{ji}$, der M_{ji} er elementet som befinner seg i rad j, kolonne i i M-matrisen. Projeksjonen $p(s, \theta)$ til dette legemet med innstrålingsvinkel $\theta = 0$ er gitt ved

$$p(m, 0) = \sum_{j=1}^{3} f_{mj} = \sum_{j=1}^{3} M_{jm}, \tag{7}$$

for kolonne nummer m. Med $\theta = \pi/2$ er projeksjonen gitt ved

$$p(m, \pi/2) = \sum_{i=1}^{3} f_{im} = \sum_{i=1}^{3} M_{mi}, \tag{8}$$

for rad nummer m. Disse projeksjonsverdier for legemet gitt av matrisen i (6) er gitt i tabell 1.

Tabell 1: Projeksjonsverdier

m	1	2	3
$p(m, 0)$	0	1	0
$p(m, \pi/2)$	0	1	0

Tilbakeprojeksjonen $g(x, y)$ kan så beregnes ved å addere alle verdier for $g(x, y; m, \theta)$, der x og y holdes fast, og $g(x, y; m, \theta)$ er gitt ved

$$g(x, y; m, \theta) = \begin{cases} p(x, \theta) & \text{for } \theta = 0 \\ p(y, \theta) & \text{for } \theta = \pi/2 \end{cases}. \tag{9}$$

Ved å summere slik og normere slik at $\sum_{ij} g(x_i, y_i) \equiv \sum_{ij} g_{ij} = 1$, oppnås følgende uttrykk for tilbakeprojeksjonen

$$g(x, y) = \frac{1}{6}[p(x, 0) + p(y, \pi/2)]. \tag{10}$$

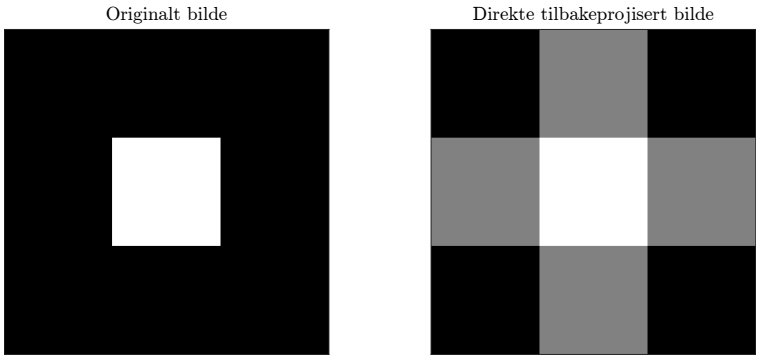
I numerisk matriseform blir dette

$$N = \begin{bmatrix} 0 & 1/6 & 0 \\ 1/6 & 1/3 & 1/6 \\ 0 & 1/6 & 0 \end{bmatrix}. \tag{11}$$

Det er derfor en betydelig skarphetsreduksjon i tilbakeprojeksjon ift. det originale bildet. En visualering av dette resultatet presenteres i oppgave 4.

4 Oppgave 4

Beregningene foretatt i oppgave 3 kan implementeres i `MATLAB` og deretter visualiseres med `imagesc()`-funksjonen. Skriptet `task4.m` gjør nettopp dette. Det originale bildet av et kvadratisk bein i et kvadratisk legeme av mykt bein fra oppgave 3, og dets tilbakeprojeksjon, visualuseres i figur 1.



Figur 1: 3×3 tilbakeprojisering (fra 6 projeksjoner) av et kvadratisk bein i et kvadratisk legeme bestående av mykt vev.

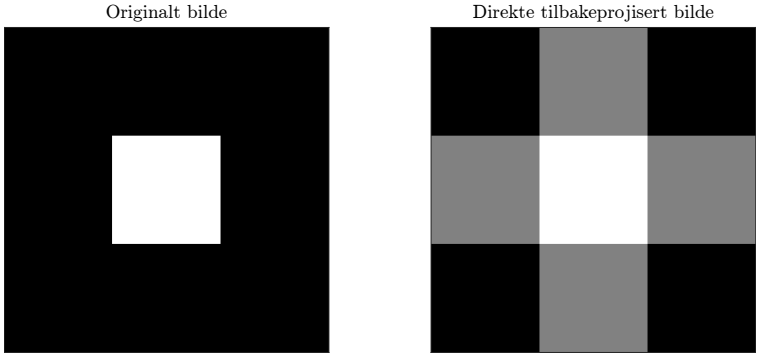
Det er nå tydelig at det blir innført en betraktelig reduksjon i bildets skarphet ved tilbakeprojeksjon.

5 Oppgave 5

Det er naturlig å anta at inkludering av flere projeksjoner, vil føre til et mindre kvalitetstap. Derfor inkluderes nå diagonale projeksjoner, dvs. $\theta = \pi/4$ og $3\pi/4$. Hver projeksjon er en sum av tre matriseelemnter f_{ij}, hvor f.eks. f_{11}, f_{12}, og f_{21} er summert i en av projeksjonene for $\theta = 3\pi/4$. Disse projeksjonene blir beregnet av `task5.m` og tilbakeprojiseres på likt vis som i oppgave 4. Tilbakeprojeksjonsmatrisen har nå følgende verdier

$$N^* = \begin{bmatrix} 1/12 & 1/8 & 1/12 \\ 1/8 & 1/6 & 1/8 \\ 1/12 & 1/8 & 1/12 \end{bmatrix}. \tag{12}$$

Sammenlignet med resultatet i oppgave 3 (ligning 11), ser det ut til at verdiene $\sum f_{ij}$ er nå jevnere fordelt utover hele matrisen. Man vil trolig anta at dette vil føre til en kvalitativt dårligere visualisert tilbakeprojeksjon sammenlignet med det originale bildet. Visualisering av N^*-matrisen med `imagesc()`-funksjonen, vist i figur 2, ser dog ut til å være identisk med visualiseringen av N-matrisen presentert i figur 1.



Figur 2: 3×3 tilbakeprojisering (fra 12 projeksjoner) av et kvadratisk bein i et kvadratisk legeme bestående av mykt vev.

Årsaken til dette er at funksjonskallet `imagesc(N); colormap('gray')` skalerer N-matrisen slik at *hele* gråskalaspekteret benyttes i visualiseringen. Resultatet vil derfor være identisk med følgende funksjonskall `image(3*N); colormap('gray')`, hvor `image()`-funksjonen er identisk med `imagec()`, bare at den ikke skalerer, og `N` er matrisen fra (11). Ved bruk av `imagesc()`-funksjonen er det derfor en implisitt antakelse om at bildet som gjenskapes har minst en helt hvit og en helt svart piksel. Det viser seg at N og N^* er ekvivalente etter en slik skalering, og visualiseres derfor på identisk vis. Dersom ingen informasjon er tilgjengelig om originalbildet, og kun visualiseringen av tilbakeprojeksjonen tas i betraktning, er det derfor ikke tilstrekkelig grunnlag til å fatte beslutninger om de absolutte numeriske verdiene f_{ij} i det originale bildet. Skaleringen fører til at kun pikslenes *relative* verdi ift. originalbildets pikselverdibredde kan infereres.

Tilbakeprojeksjonens numeriske nøyaktighet kan uttrykkes ved RMS-avviket fra originalbildets numeriske verdier. N og N^* har henholdsvis 0.25 og 0.30 i RMS-avvik, noe som indikerer at ekstra projeksjoner *ikke* har ført til et bedre numerisk resultat for tilbakeprojeksjonen i dette tilfellet.

6 Oppgave 6

Funksjonen `getProjection()` beregner de $2N - 1$ projeksjonene av et bilde $f(x, y)$ med $N \times N$ piksler, for en gitt vinkel $0 \leq \theta < 180°$. I et kartesisk koordinatsystem har man piksel $(i, j) = (x_i, y_j) = ((2i - N - 1), (2j - N - 1)) \cdot a/2$

(der a er bredden til pikslene) og hvor $1 \leq i, j \leq N$. s-verdien for en enkelt projeksjonen m, $s_m = -(N-m)a/\sqrt{2}$, gir hvor røntgenstrålen treffer bildet ($1 \leq m \leq 2N-1$). s-verdien til hver enkelt piksel beregnes med følgende formel:

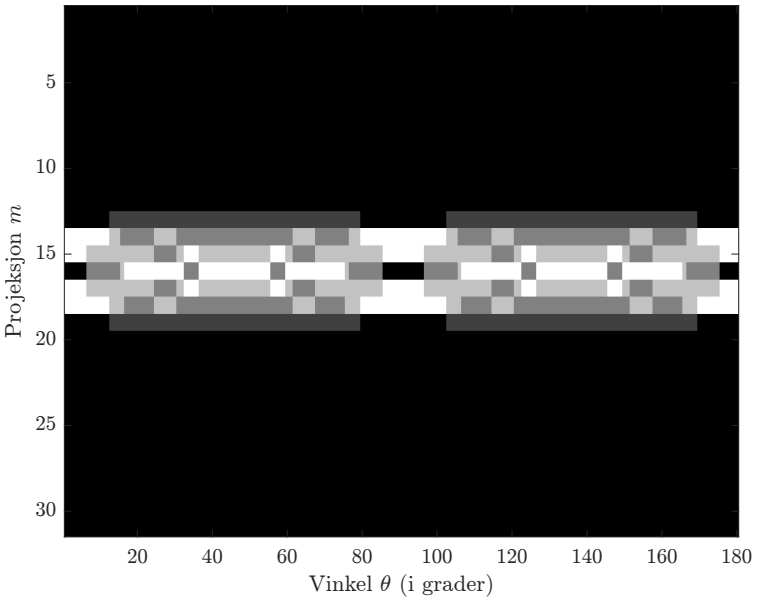
$$s = x\cos(\theta) + y\sin(\theta), \tag{13}$$

som gitt i oppgaven. Avstanden mellom hver projeksjon er $a/\sqrt{2}$. Ved å bruke formelen

$$m = \frac{s}{a/\sqrt{2}} + N, \tag{14}$$

og runde av til nærmeste heltall, finner man hvilken projeksjon en piksel hører til. Bredden a kanselleres derfor ved innsetting, og den ses derfor bort fra (gis verdien 1) i funksjonene.

`getSinogram()` benytter funksjonen `getProjection()` til å beregne projeksjonene for hver av et gitt antall uniformt fordelte vinkler n_θ, og returnerer dem i $\{p(s,\theta)\}$, en $(2N-1) \times n_\theta$ matrise. `saveSinogramTask6.m` kjører `getSinogram()` med $n_\theta = 180$ på et bilde med 16×16 piksler, der de 4×4 i sentrum av bildet er hvite, og resten sorte. Sinogrammet lagres så i en tekstfil. `task6.m` laster inn denne filen, og tegner opp sinogrammet vist i figur 3.

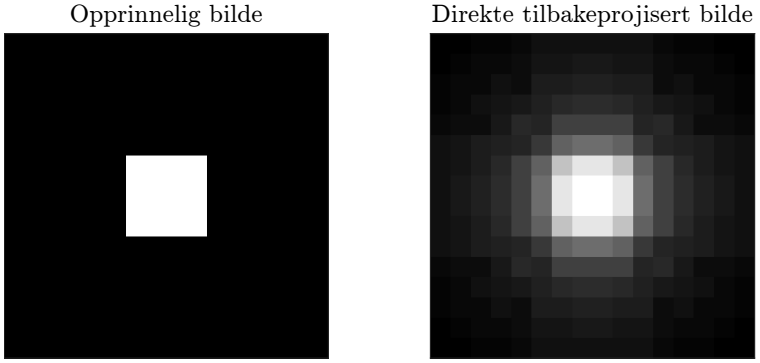


Figur 3: Sinogrammet $p(s,\theta)$ til et bilde med 16×16 piksler, der de 4×4 i midten er hvite og resten sorte, med antall vinkelverdier $n_\theta = 180$.

7 Oppgave 7

For alle bilder med samme antall piksler $N \times N$, vil alltid en bestemt piksel (i, j) høre til samme projeksjon m for en gitt vinkel θ. I `saveNewProjectionsPixelsArray()` benyttes dette ved å lagre en $(2N-1) \times n_\theta$ cellematrise som lagres i en `.mat`-fil. Hver celle i denne cellematrisen består av en punktliste (en matrise med to kolonner) som inneholder hvilke piksler som hører til en gitt projeksjon. Samme framgangsmåte som beskrevet i oppgave 6 brukes for å allokere pikslene til riktig projeksjon. `getProjectionsPixelsArray()` laster inn en cellematrise av nevnte type fra en fil for gitte N- og n_θ-verdier. Dersom det ikke finnes en lagret matrise for disse verdiene, kjøres først `saveNewProjectionsPixelsArray()`.

`getBackProjection()` initialiserer en $N \times N$ nullmatrise. Deretter går den gjennom alle projeksjonene, med verdi p_{mn}, og legger til verdien p_{mn}/M til projeksjonens tilhørende piksler, der M er antallet piksler som hører til projeksjonen. Resultatet er et tilbakeprojisert bilde $g(x, y)$. I `task7.m` tegnes tilbakeprojeksjonen gitt av sinogrammet beregnet i oppgave 6, ved siden av det opprinnelige bildet. Resultatet er gitt i figur 4.



Figur 4: 16×16 pikslers bilde med tilhørende tilbakeprojeksjon fra sinogram beregnet med antall piksler $n_\theta = 180$.

Det er tydelig at de hvite pikslene "smøres" en del utover, spesielt langs vannrette og loddrette linjer som går tilnærmet gjennom sentrum av bildet. Dette er som forventet, siden flere projeksjoner med verdi større enn null går gjennom de mer sentrale pikslene.

Som nevnt i oppgave 5, bruker `imagesc()` hele gråskalaen. Dette er ikke

noe problem i seg selv, bare det er bevissthet rundt at helt hvite piksler har den høyeste verdien i tilbakeprojeksjonen, mens helt svarte har den laveste verdien. Generelt korrelerer disse (dersom tilbakeprojeksjonen er god) med hhv. de lyseste og de mørkeste områdene i objektet man ønsker å fremstille. For å regne ut et meningsfylt RMS-avvik mellom verdiene i tilbakeprojeksjonen og det originale bildet, må det foretas en skalering av verdiene i førstnevnte. Pikslene i det originale bildet er gitt verdiene $0 \leq g(x, y) \leq 1$ (de har verdiene 0 eller 1), og vi ønsker således at det samme skal gjelde pikslene i tilbakeprojeksjonen. Dette oppnås enkelt ved å trekke verdien av det laveste elementet i matrisen som representerer tilbakeprojeksjonen fra alle elementene, og deretter dele på (den nye) verdien av det største elementet. Disse operasjonene vil ikke endre på den visuelle fremstillingen av tilbakeprojeksjonen. RMS-avviket beregnet med denne skaleringen vil være korrelert til kvaliteten til den visuelle fremstillingen av objektet.

`task6.m` Regner ut RMS-avviket med metoden beskrevet over. $n_\theta = 180$ gir RMS ≈ 0.1457, mens $n_\theta = 360$ gir RMS ≈ 0.1444, altså ikke store forskjellen. For pikselverdier mellom 0 og 1 (0 eller 1, i dette tilfellet), er det uansett et betydelig avvik. Tilbakeprojeksjonen er allikevel tydelig nok til at man får et godt inntrykk av hva som fremstilles i det opprinnelige bildet.

8 Oppgave 8

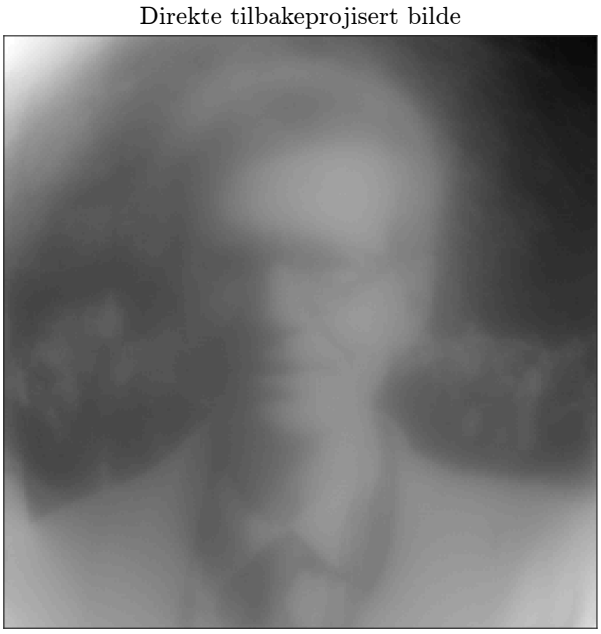
`task8.m` benytter funksjonene beskrevet i oppgave 7 til å rekonstruere bildet $f(x, y)$, bestående av 256×256 piksler, med sinogram gitt i **sinogram01.txt**, der $n_\theta = 180$. Deretter lagres tilbakeprojeksjonen til fil, slik at denne kan brukes i oppgave 9 (det samme er gjort med standardbildet *phantom*). Tilbakeprojeksjonen $g(x, y)$ er vist i figur 5.

9 Oppgave 9

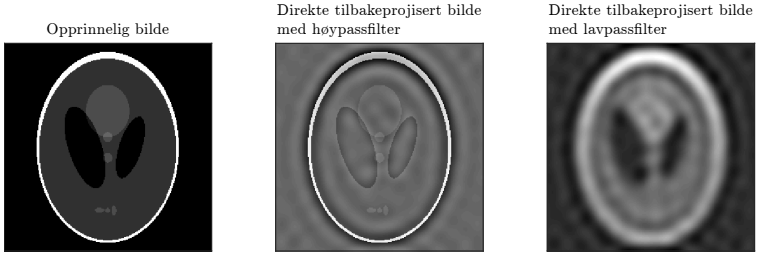
Ved å anvende først et høypassfilter og deretter et lavpassfilter på sinogrammet fra Phantom-figuren i Matlab, fjernes henholdsvis de lave og de høye Fourierkomponentene (frekvensene) fra den Fouriertransformerte av bildet. Gitt Phantom-bildet med 256×256 piksler, fås figur 6 ved å sette alle pikslene innenfor en radius lik 10 piksler fra sentrum av bildet lik 0 for høypassfilteret, og alle pikslene utenfor en radius lik 10 piksler fra sentrum av bildet lik 0 for lavpassfilteret.

For visuelt å kunne bekrefte at tilbakeprojeksjonen av et bilde er en uskarp versjon av det originale bildet, kan bildet Phantom (igjen med 256×256 piksler) og dets tilbakeprojeksjon brukes. Først beregnes de Fouriertransformerte av bildet f, og dets tilbakeprojeksjon g. Dernest tas logaritmen til absoluttverdien til de Fouriertransformerte matrisene og plottes ved hjelp av `imagesc()` i Matlab. Resultatet kan ses i figur 7.

Området med gulfarge i hjørnene på figuren til venstre er tydelig klarere enn

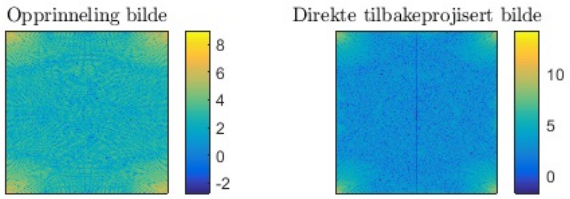


Figur 5: 256×256 pikslers tilbakeprojeksjon av bilde med sinogram gitt i **sinogram01.txt**, som er beregnet med antall piksler $n_\theta = 180$.



Figur 6: 256×256 piksler-versjonen av Phantom-bildet fra Matlab, først tilbakeprojisert med et høypassfilter og deretter med et lavpassfilter.

tilsvarende i den til venstre. Det er skarpere fargekontraster i det opprinnelige bildet, enn i den tilbakeprojiserte figuren. Dette stemmer godt overens med at g er en uskarp utgave av f.

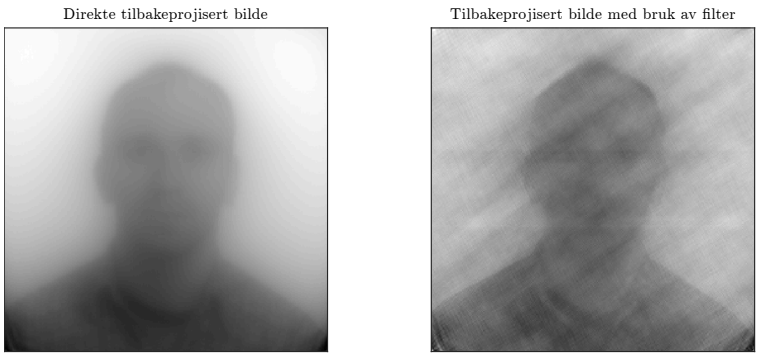


Figur 7: Plot av logaritmen til absoluttverdien av den Fouriertransformerte til 256×256 piksler-versjonen av Phantom-bildet fra Matlab til venstre, samt tilsvarende for den direkte tilbakeprojiserte til høyre.

En fremgangsmåte for å gjøre tilbakeprojeksjonen g skarpere er å lage et filter (en matrise) i Fourierrommet som multipliseres elementvis med den Fourier-transformerte av tilbakeprojeksjonen, før denne inverstransformeres tilbake.

Den første fremgangsmåten består i å laste ned et portrettbilde, og lagre dette i svart-hvitt-versjon med 256×256 piksler. Tankegangen bak å bruke et portrettbilde er at sinogrammene som ble tildelt viser seg å også være portrett-bilder. Filteret lages ved å Fouriertransformere selve bildet, samt den direkte tilbakeprojiserte versjonen, for så å dele matrisene elementvis på hverandre; altså originalen delt på tilbakeprojeksjonen (i Fourierrommet). Denne matrisen, A, multipliseres så elementvis med tilbakeprojeksjonen beregnet fra det utdelte sinogrammet. Resultatet kan ses i figur 8. Enkelte detaljer er ikke lenger like tydelige, men overgangene (dvs. konturene) er skarpere.

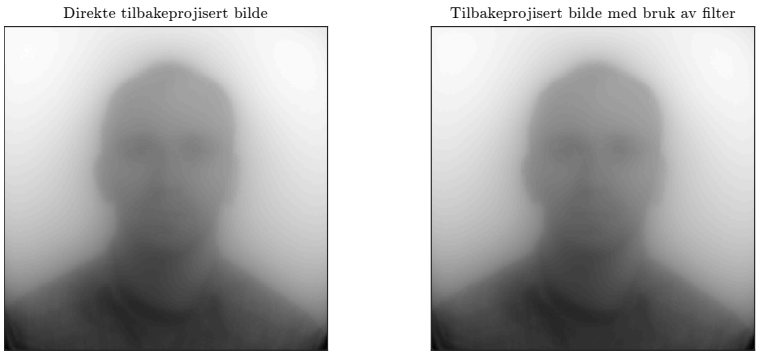
Et nytt filter beregnes med en annen fremgangsmåte. Denne gangen tas en 256×256-matrise med alle elementer innenfor en gitt radius fra sentrum lik 0. Fra denne radiusen til en radius lik 128 øker verdiene til elementene lineært fra 0 til 1, og de resterende verdiene i hjørnesegmentene settes lik 1. Tankegangen bak denne fremgangsmåten er at detaljene i midten av bildet skulle fremheves bedre, da det er her de fleste detaljene ser ut til å ligge. Denne matrisen, en annen versjon av A, multipliseres så, på tilsvarende måte, elementvis med



Figur 8: Den direkte tilbake-projiserte versjonen av bildet til venstre, og tilsvarende med bruk av filter laget med et annet portrettbilde til høyre.

tilbakeprojeksjonen av det bildet vi var blitt gitt sinogrammet til. Resultatet kan ses i figur 9. Her er bildene nær sagt identiske - men ved å forstørre bildene kan man se at bildet der filteret er brukt er marginalt bedre.

Det er tydelig at ingen av filtrene gjorde figuren bedre i noen særlig grad. Likevel fremheves enkelte detaljer bedre enn en direkte tilbakeprojeksjon klarer alene.



Figur 9: Den direkte tilbake-projiserte versjonen av bildet til venstre, og tilsvarende med bruk av filter med høyere og høyere verdier fra 0 og opp til 1 til høyre.