NORGES HANDELSHØYSKOLE
Bergen, våren 2007

Utredning i fordypnings-/spesialfagsområdet: Bedriftsøkonomisk analyse
Veileder: Professor Kurt Jørnsten

Revenue Management -

av
Bjørn Larsen

Forord

Denne masteroppgaven ble utført ved Norges Handelshøgskole i Bergen ved institutt for foretaksøkonomi, og ble avsluttet i juni 2007.

Masteroppgaven tar for seg revenue management innen hotellindustrien og ser på forskjellen i lønnsomheten ved bruk av metodene EMSRA, EMSRB og dynamisk programmering.

Jeg vil gjerne takke proffesor Kurt Jørnsten for gode råd og veiledning underveis.

Bergen, juni 2007

Bjørn Larsen

Innhold

Innledning

I denne masteroppgaven har jeg tatt for meg revenue management innen hotellindustrien.
I oppgaven er det gjort rede for ulike metoder som kan brukes for å avgjøre om hotellet skal akseptere en forespørsel for en prisklasse eller ikke og lønnsomheten til disse metodene. For å kunne se på lønnsomheten til de ulike metodene har jeg laget et dataprogram. Dataprogrammet tar for seg et hotell hvor det kun er mulig å få inn forespørsler for en natt. Programmet er laget i programmeringsspråket Java og en god del av arbeidet med oppgaven har bestått i å lage dette programmet. Siden dette er en oppgave i økonomi, har jeg i oppgaveteksten ikke lagt så mye vekt på å forklare de programmeringsmessige utfordringene i programmet, men lagt vekt på det matematiske og økonomiske. Jeg forklarer hvordan programmet virker matematisk, uten å legge så mye vekt på hvordan dette er oppnådd programmeringsmessig. Oppgaven starter med en problembeskrivelse, hva oppgaven tar for seg og hva den ikke tar for seg. Jeg fortsetter så med et lite stykke om Javaprogrammering. Dette stykket er ikke så langt, men nok til at en kan få et lite innblikk i programmeringsspråket Java. Jeg gir deretter en kort innføring i hva Revenue management er, hvilke bedrifter som kan bruke det og historien til revenue management i flyindustrien. Grunnen til at jeg tar for meg historien til revenue management i flyindustrien er at det var der revenue management startet. For at metodene skal kunne regne ut hvilke forespørsler som skal aksepteres og hvilke som ikke skal aksepteres, må de ha et forecast. I kapittel 3 tar jeg for meg hvordan vi kan komme frem til et slik forecast og andre problemstillinger rundt dette temaet. Jeg går så gjennom teorien til de ulike metodene som blir brukt i dataprogrammet, for å regne ut om en forespørsel skal akseptere eller ikke. Dette er metodene EMSRA, EMSRB og dynamisk programmering. Jeg har også en teoretisk gjennomgang av Popescu og Bertsimas sin teori om Certainty equivalent control (cec) selv om denne ikke er brukt i dataprogrammet. Denne teorien kan brukes når det er mulig å bestille rom for flere netter. Etter at jeg har tatt for meg

teorien, gjennomgår jeg programmet i forhold til hvordan dette er laget og fungerer. Til slutt foretar jeg noen tester for å få innblikk i metodene, og kommer med en oppsummering på hvordan det gikk.

Oppgavebeskrivelse

Denne oppgaven gjør rede for revenue management innen hotellindustrien. Hovedformålet i oppgaven er å se på forskjeller i lønnsomhet ved bruk av ulike metoder for å avgjøre om hotellet skal akseptere en forespørsel for en prisklasse eller ikke. Metodene jeg har sett på er EMSRa, EMSRb og dynamisk programmering. For å kunne vurdere lønnsomheten til de ulike metodene, har jeg laget et dataprogram i Java. Dette programmet stipulerer en tilfeldig etterspørsel ut i fra den forventede etterspørselen til hotellet, og regner ut lønnsomheten til de forskjellige metodene med den stipulerte etterspørselen. Jeg har tatt utgangspunkt i et imaginært flyplasshotell, for å kunne vurdere lønnsomheten til de ulike metodene. Jeg går ut fra at hotellet kun mottar forespørsler for overnattinger på 1 natt. Dette er en akseptabel forutsetning, ettersom de fleste forespørsler på et flyplasshotell er for en natt. For mer om dette, se Ben Vinod.
I mangel på konkret informasjon om etterspørselsmønsteret til hotellet, har jeg istedetfor tenkt ut en forventet etterspørsel i denne oppgaven. Den forventede etterspørselen har jeg kommet frem til ved hjelp av informasjon jeg har fått av Line Akerlind (Thon) og Tomm Caspersen (Rica).
I denne oppgaven er det ikke tatt hensyn til overbooking. Overbooking er når hotellene leier ut flere rom enn de har. Dette gjøres fordi hotellet regner med at det kommer inn avbestillinger eller at noen ikke møter opp. Ved å ikke overbooke risikerer hotellet å ha tomme rom som kunne ha vært leid ut til andre kunder. Møter det flere kunder opp enn det er rom til på hotellet (på grunn av overbooking) blir en nødt til å avvise noen eller finne alternativ overnatting. Å finne rett overbookingsnivå er derfor en balansegang mellom kostnaden av å ha tomme rom som kunne ha vært leid ut, mot kostnaden av å måte avvise kunder som har bestilt rom. Jeg har altså gått ut fra at hotellet ikke får avbestillinger og at alle som har bestilt rom bruker det.
Tiden mellom hver forespørsel etter å leie rom, forutsetter jeg at kan beskrives av en poisson prosess. Det blir ikke tatt hensyn til muligheten for trade up. Trade up vil si at kunden som ønsker å leie rom, leier et dyrere rom når hotellet er utsolgt for den prisklassen kunden ønsker. Det blir ikke tatt hensyn til at kunden kanskje leier rom på en annen dag hvis hotellet er fullt.
Det blir også gått ut fra at de som bestiller rom kun bestiller ett rom, og det blir dermed sett vekk fra bedrifter som bestiller flere rom i forbindelse med konferanser og feriereisende som bestiller flere rom.

Kapittel 1

I denne oppgaven gjør jeg bruk av et dataprogram som er programmert i programmeringsspråket Java. Det vil i dette kapitlet bli gitt en kort innføring i dette programmeringsspråket.

Programmeringsspråket Java

Java er et objektorientert programmeringsspråk som ble utviklet av Gosling, Naughton, Warth, Frank og Sheridan i Sun Microsystems i 1991. Det tok 18 måneder å utvikle den første arbeidsversjonen. Denne ble kalt Oak og fikk senere navnet Java. Java ble laget for at det skulle være lett å lære å bruke for profesjonelle programmerere og har derfor mange likheter med programmeringsspråket C++.

Det er to hovedtyper av programstrukturer; prosessorientert og objektorientert. Java er som tidligere nevnt i oppgaven et objektorientert programmeringsspråk. En av fordelene med objektorientert programmering i forhold til prosessorientert programmering, er at det gjør programmereren i stand til å lage moduler som ikke trengs å endres når et nytt objekt blir lagt til. Programmereren kan lett lage et objekt som arver mange av sine egenskaper fra eksisterende objekt. Dette gjør objektorientert programmering lettere å modifisere.

Et objekt kan for eksempel være en hest, en sko eller en bil, eller som i denne oppgaven et bestemt forespørselsmønster som blir stipulert i stipuleringsklassen. For å lage et objekt må en ha en klasse som består av en eller flere metoder og variabler. Hver gang programmet kjører en klasse vil et nytt objekt bli laget. Når for eksempel stipuleringsklassen blir kjørt, dannes et nytt forespørselsobjekt (et nytt forespørselsmønster). Alle forespørselsmønstre tilhører stipuleringsklassen, samtidig som de er egne objekt. Metodene i klassen kan en se på som handlinger som objektet kan utføre, mens variablene er egenskapene til objektet. Alle objekt av

samme klasse har samme egenskaper, men de har ikke de samme egenskapsverdiene(Naughton & Schildt, 1999).

Kapittel 2

I dette kapitlet gjør jeg rede for hva revenue management er, og gir et historisk eksempel på bruk av revenue management i flyindustrien. Eksemplet fra flyindustrien er valgt fordi det var innenfor denne industrien revenue management utviklet seg i starten. Dette kapitlet vil gi leseren et innblikk i hva revenue management er og hvor effektivt det kan være i bruk.

Revenue Management

Revenue management er en term som blir brukt for å beskrive prosessen med å oppnå maksimal inntekt fra salget av ”perishable” ressurser. Utfordringen ligger i å selge den riktige ressursen til den riktige kunden på det riktige tidspunktet. Dette skjer i en kombinasjon mellom det å dele markedet opp i flere markedssegmenter, styre ledige ressurser (f.eks. hotellrom eller flyseter), estimering av fremtidige forespørsler samt prising. Revenue management startet i flyindustrien og har senere spredt seg til en rekke andre næringer, som for eksempel bilutleie, hotelldrift og salg av charterturer. Følgende felles karaktertrekk gjelder for virksomheter som bruker eller er egnet til å bruke revenue management:

1. Det er dyrt eller umulig å lagre overskuddskapasitet
2. Fast kapasitet (på kort sikt)
3. Høye faste kostnader
4. Lave marginale kostnader
5. Firma kan skille mellom ulike kundesegment, og hvert kundesegment har forskjellig etterspørselskurve
6. Samme enhet av kapasitet kan brukes til å levere mange forskjellige produkter eller tjenester

Revenue management fokuserer på å maksimere forventet marginal inntekt for en gitt operasjon og planleggingsperiode. Revenue management optimaliserer ressursutnyttelsen ved å sikre ressurstilgjengeligheten til de kundene som har den høyeste forventede betalingsviljen.

Starten på Revenue management

Starten på revenue management mener de fleste var i slutten av 70-årene da det amerikanske flymarkedet ble deregulert. Dr. Garette van Ryzin mener imidlertid at revenue management ikke er en ny ide, men noe økonomer har tenkt på også i fra gammelt av. Det som er nytt er bare hvordan beslutningene blir tatt; med bruk av informasjonsteknologi og vitenskaplig beslutningstaking. Følgende sitat fra Jules Dupuit, som var en fransk sivilingeniør og økonom, er et eksempel på dette:

"It is not because of the few thousand francs which would have to be spent to put a roof over the third-class carriages or to upholster the third-class seats that some company or other has open carriages with wooden benches … What the company is trying to do is prevent the passengers who can pay the second-class fare from travelling third -class; it hits the poor, not because it wants to hurt them, but to frighten the rich."

Eksemplet som er hentet fra prising av togbilletter, viser hvordan en prøvde å skille passasjerer etter betalingsviljen, og er således et eksempel på bruk av revenue management i tidligere tider.

Skal en se på historien til revenue management, er det imidlertid naturlig å starte med å se på flyindustrien sin utvikling av revenue management. I 1953 ble den første kommersielle computeren laget i flere eksemplarer, produsert av IBM. Samme år ble også det første computerbaserte reservasjonssystemet påbegynt. Dette kom i stand i et møte mellom IBM og American Airlines. Prosjektet ble kalt Saber og skiftet senere navn til Sabre. Sabre kom online i 1962, og American Airlines ble i påfølgende år etterfulgt av flere av de store flyselskapene. En viktig årsak til at flyselskapene kan bruke revenue management så effektivt er informasjonen de får fra disse reservasjonssystemene. Reservasjonssystemene teller ikke bare antall seter som er

solgt, men teller også antall seter solgt i hver prisklasse. Dette gjør at flyselskapene kan regne ut sannsynligheten for hvor mange seter de kan selge i hver prisklasse.

Før dereguleringen hadde flyselskapene relativt liten anledning til å ta forskjellige priser fra forskjellige kunder på den samme flyavgangen. Regulert ved lov, betalte hver passasjer på hvert fly den samme prisen for å komme fra by A til by B. Det var noen unntak; flyselskapene kunne for eksempel tilby spesielle red-eye priser og tilbud til studenter. Reservasjonssystemene var derfor laget slik at de kunne ta hensyn til forskjellige prisklasser.

Etter dereguleringen ble selskapene friere til å sette priser og det ble åpnet opp for konkurranse med fri etablering og utgang fra markedene. Dette førte til etablering av lavkostnadsselskaper og hard konkurranse for de etablerte selskapene. Et eksempel på dette er PeopleExpress som startet i 1981, og som innen 1984 hadde inntekter på 1 milliard dollar og 60 millioner dollar i fortjeneste. Suksessen til lavprisselskapene hadde (selvfølgelig) stor innvirkning på de store selskapene. Bob Crandall i VP marketing for American Airlines, la i denne perioden merke til noen essensielle fakta:

1. Mange American Airlines fly fløy med ledige seter
2. Marginal kostnaden ved å bruke disse var veldig liten
3. American Airlines kunne konkurrere på kostnader ved å bruke sine ”overskuddsseter”

Disse oppdagelsene førte til at DINAMO (Dynamic Inventory Allocation and Maintenance Optimizer) ble startet av American Airlines. American Airlines laget nye tilbudspriser, med forskjellige restriksjoner (“Super Saver” and “Ultimate Super Saver” fares) og lanserte dem med brask og bram i januar 1985. Flyanalytikere trodde dette var starten på en priskrig: “American cannot operate profitably at these fares.” Men de nye prisklassene var American Airlines metode for å segmentere markedet og utnytte de ledige setene. DINAMO viste seg å være meget effektivt, noe som gikk ut over konkurrentene. Peoples Express gikk fra å ha 60 millioner dollar i fortjeneste i 1984 til å få et tap i 1985 på 160 millioner dollar, og innen 1986 var firmaet konkurs og solgt til Continental. Her er et sitat fra Donald Burr, CEO of PeopleExpress:

"We were a vibrant, profitable company from 1981 to 1985, and then we tipped right over into losing $50 million a month. We were still the same company. What changed was American's ability to do widespread Yield Management in every one of our markets. We had been profitable from the day we started until American came at us with Ultimate Super Savers. That was the end of our run because they were able to under-price us at will and surreptitiously."

"Obviously PeopleExpress failed . . . We did a lot of things right. But we didn't get our hands around Yield Management and automation issues. [If I were to do it again . . .] the number one priority on my list every day would be to see that my people got the best information technology tools. In my view, that's what drives airline revenues today more than any other factor–more than service, more than planes, more than routes."

Dette eksemplet på konkurransen mellom American Airlines og PeopleExpress, viser hvor effektivt revenue management kan være. American Airlines vant i konkurransen med PeopleExpress ved å bruke informasjonen de fikk fra reservasjonssystemet til å utvikle en strategi i revenue management (Bollapragada, 2005).

Kapittel 3

Etterspørsel og forecast

I dette kapitlet tar jeg for meg to forskjellige etterspørselstyper, spiraldown og hvordan en kan komme frem til et forecast til bruk i revenue management modellene.

Etterspørsel

Et hotell kan ha to typer etterspørsel eller det kan ha en kombinasjon av disse to. De to etterspørselstypene er yieldable og priceable etterspørsel. I denne oppgaven går jeg ut fra at hotellet kun har yieldable etterspørsel. Dette går jeg ut fra fordi metodene jeg har brukt, er basert på yieldable etterspørsel. Jeg vil derfor først gjennomgå hva som menes med yiealdable etterspørsel. Yiealdable etterspørsel har vi når kunden kun er interessert i å leie rom i en bestemt prisklasse. Er ikke denne prisklassen tilgjenglig, leier han ikke rom. Et eksempel på yieldable etterspørsel kan sees i tabell 1. I den første raden til tabellen får en oppgitt kundens maksimale betalingsvilje, den er enten høy (H) eller lav (L). Hvor H er prisen på den dyreste prisklassen og L er prisen på den billigste prisklassen. I rad to har enn de prisklassene som er tilgjengelige for kunden og i rad tre er den prisklassen kunden velger å leie rom i. I den første linjen har vi en kunde som har en betalingsvilje på H, og som kan velge mellom å leie rom i prisklassene H eller L. Siden en har yiealdable etterspørsel ønsker han ikke å leie rom i prisklasse L selv om denne er billigere enn prisklasse H. Han leier derfor rom i prisklasse H. I den andre linjen har vi en kunde som har en betalingsvilje på L og som kan velge mellom rom i prisklasse H eller L. Siden hans maksimale betalingsvilje er L velger han å leie rom i prisklasse L. I den tredje linjen har en en kunde som har betalingsvilje på H, og som kun kan velge å leie rom i prisklasse H. Ettersom prisklasse H ikke overstiger hans betalingsvilje og det er denne prisklassen han ønsker å leie rom i, leier han rom i prisklasse H. I den fjerde linjen har vi en kunde som har en betalingsvilje på L, og som kun kan velge å leie rom i prisklasse H. Da prisklasse H er over betalingsviljen til kunden, velger han å ikke leie rom.

Tabell 1 Yiealdable etterspørsel

Kunde etterspørsel	Tilgjengelige klasser	Booking resultat
H	H og L	H
L	H og L	L
H	Kun H	H
L	Kun H	Ingen

På et hotell er mesteparten av rommene forholdsvis like og i denne oppgaven har jeg gått ut fra at alle rommene på hotellet er like. I utgangspunktet er det altså uten betydning for kunden hvilket rom han leier, og i en slik situasjon vil kunden velge den billigste prisklassen som er tilgjengelig. For at hotellet skal kunne ha en yiealdable etterspørsel må derfor hotellet skille de ulike prisklassene fra hverandre. Dette kan gjøres ved hjelp av ulike restriksjoner på de billigste prisklassene, ved å yte noe ekstra til de dyreste prisklassene eller ved at de kun lar kunden se den prisklassen de ønsker at han skal leie i. Restriksjoner på å få leie i en bestemt prisklasse kan for eksempel være begrensninger i mulighet for å avbestille rommet, tidspunkt for innsjekking, minimum antall dager en må bo på hotellet eller at en må tilhøre en bestemt gruppe for å få tilbudet o.s.v.

Ved priceable etterspørsel vil kunden velge den billigste prisklassen som er åpen for utleie, hvis den ikke er dyrere enn hans betalingsvilje. Et eksempel på Priceable etterspørsel ser vi i tabell 2. Sammenligner vi tabell 1 og 2 ser vi at den eneste plassen en priceable kunde velger forskjellig fra en yiealdable kunde er i linje 1. I linje 1 har vi en kunde med maks betalingsvilje på H. Siden vi har en kunde med priceable etterspørsel vil han velge å leie et rom i den billigste prisklassen som er tilgjengelig. Han har valget mellom prisklasse H eller L, hvor han vil velge å leie i prisklasse L siden det er den billigste prisklassen.

Tabell 2
Priceable etterspørsel

Kunde etterspørsel	Tilgjengelige klasser	Booking resultat
H	H og L	L
L	H og L	L
H	Kun H	H
L	Kun H	Ingen

Priceable etterspørsel har vi når det ikke er noen restriksjoner på rommene eller restriksjonene er uten betydning for kunden. Ved en priceable etterspørsel vil etterspørselen etter de billigste rommene alltid være større eller lik etterspørselen etter de dyreste rommene, ettersom en som er villig til å betale en høy pris alltid vil leie et billigere rom dersom det er tilgjengelig.

De to forskjellige typene etterspørsel oppfører seg forskjellig, og har behov for forskjellige forecasting og optimaliseringsmetoder.

Generelt vil det gjelde at dersom vi bruker yieldable etterspørsel når vi har priceable etterspørsel, vil et forecast overestimere etterspørselen etter de billigste prisklassene på bekostning av etterspørselen etter de dyreste prisklassene (Et forcast er i dette tilfellet et estimat på forventet fremtidig etterspørsel etter å leie rom på hotellet, og vil bli nærmere forklart senere i oppgaven). Dette vil føre til at for få rom blir reservert til de dyreste prisklassene, noe som fører til tap av inntekter. Hvis prosessen fortsetter fra en forecasting periode til en annen, vil vi kunne oppleve en spiral down effekt (hva som menes med spiral down effekt vil bli forklart senere i kapitlet).

Selv om vi skulle klare å få et riktig forecast, vil ikke en yiealdebale prismodell være optimal når vi har priceable etterspørsel. Dette skyldes at en yiealdable løsningsmetode går ut fra at de som vil leie rom i den dyreste prisklassen kun leier rom i denne prisklassen, men ved priceable etterspørsel vil de velge å leie rom i den billigste prisklassen som er tilgjengelig. Løsningsmetoden kan derfor føre til at de billigste prisklassene er lenger åpne for salg enn hva som er optimalt, noe som fører til buy down. Buy down er når kunden leier rom i en prisklasse som er billigere enn den prisklassen han er villig til å leie i.

Dersom forecasting metoder for priceable etterspørsel blir brukt når vi har yieadable etterspørsel, vil forecastene overestimere etterspørselen etter de dyreste prisklassene på bekostning av de billigste prisklassene. Dette vil føre til tap av inntekter som følge av at for mange rom blir reservert til de dyreste prisklassene (Boyd & Kallesen, 2004).

Spiral down

Spiral down er når feilaktige antagelser om kundenes etterspørsel fører til at salget av de dyreste prisklassene, reservasjonsnivået og inntekt systematisk blir redusert med tiden. Denne feilaktige antagelsen om kundenes etterspørsel kan oppstå når hotellet lager et forecast basert på historisk data, uten å ta skikkelig hensyn til kundenes adferd. I den påfølgende teksten har jeg gått ut fra at hotellet kun har to prisklasser: en normal pris og en tilbudspris.

Hotellet må for hvert overnattingsdøgn bestemme hvor mange rom som skal reserveres til den dyreste prisklassen. For å kunne bestemme dette reservasjonsnivået, må hotellet ha et forecast på forventet etterspørsel. Forecastet kan vi komme frem til ved hjelp av observert etterspørsel fra tidligere perioder. Men et problem med bruk av observert etterspørsel for å finne forecastet, er at den observerte etterspørselen kan være avhengig av reservasjonsnivået. Har vi priceable etterspørsel vil reservasjonsnivået ha innvirkning på observert etterspørsel etter de dyreste rommene. Reduseres reservasjonsnivået økes tilgjengeligheten av de billigste rommene. Da vil flere av de som var villig til å leie rom i den dyreste prisklassen, leie rom til tilbudspris. Den observerte etterspørselen etter de dyreste rommene blir dermed mindre. Antall kunder som hadde ønsket å leie rom i den dyreste prisklassen, hvis ikke den billigste prisklassen var tilgjengelig, er imidlertid ikke forandret. Med lavere observert etterspørsel etter den dyreste prisklassen, vil vi få et forecast som viser lavere etterspørsel etter de dyreste rommene. Reservasjonsnivået blir da satt lavere, noe som gjør at enda flere rom blir tilgjengelige til tilbudspris, som resulterer i at enda færre av de dyreste rommene blir utleid. Denne prosessen fortsetter og leder således til en spiral nedover for antall utleide rom i den dyreste prisklassen, beskyttelsesnivået og inntektene.

Antall kunder som er villige til å betale prisen for den dyreste prisklassen (hvis dette er eneste alternativ) er uavhengig av reservasjonsnivået, men den observerte etterspørselen er ofte

avhengig av reservasjonsnivået. Dette vil jeg vise ved hjelp av 4 eksempler. I det første eksemplet er den observerte etterspørselen uavhengig av reservasjonsnivået (som vi kaller for l), mens den i de 3 siste eksemplene er avhengig av reservasjonsnivået. I eksemplene er det tre typer kunder; kunde a, kunde b og kunde ab. Kunde b er kun interessert i å leie i den billigste prisklassen. Er ikke denne prisklassen tilgjenglig leier ikke denne kunden rom. Kunde a er kun interessert i å leie i den dyreste prisklassen. Selv om det er mulig å leie i den billigste prisklassen, leier vedkommende rom i den dyreste prisklassen. Kunde ab er villig til å betale prisen for den dyreste prisklassen, men hvis den billigste prisklassen er ledig velger kunden denne. Etterspørselen til kunde type a kaller vi for D_a, etterspørselen til kunde type b kaller vi for D_b og etterspørselen til kunde type ab kaller vi for D_{ab}. Vi lar $D_a(l)$ og $D_{ab}(l)$ stå for antall a og ab kunder som kommer frem til c-l rom har blitt utleid (c står for antall rom på hotellet(kapasiteten)). Antall observerte forespørsler i den dyreste prisklassen kaller vi for X. Hvis reservasjonsnivået er større eller lik kapasiteten $l \geq c$ er $D_a(l) = D_{ab}(l)=0$ og hvis den totale etterspørselen er mindre enn c-l, da er $D_a(l)=D_a$ og $D_{ab}(l)=D_{ab}$. Legg merke til at $D_a(l)$ og $D_{ab}(l)$ begge er avhengige av hvor mange forespørsler det er til hver av de tre typene med kunder og i hvilke rekkefølge forespørslene kommer i.

Eksempel 1

I dette eksemplet har vi kun type a og type b kunder og revenue manageren observerer alle forespørslene, også de forespørslene som kommer etter at vi har leid ut alle rommene. Den observerte etterspørselen i den dyreste prisklassen blir da lik etterspørselen til kundetype a ($X=D_a$). Vi lar G(l, x) stå for den kumulative distribusjonsfunksjonen til antall observerte forespørsler i den dyreste prisklassen, hvor booking prosessen er kontrollert med reservasjonsnivå l. Vi får da at:

$$G(l, x) = P[X \leq x] = P[D_a \leq x]$$

som er uavhengig av l. Det vil si at den kumulative distribusjonsfunksjonen til det observerte antallet med forespørsler er uavhengig av reservasjonsnivået. Antall forespørsler vi observerer er

ikke avhengig av reservasjonsnivået, på grunn av at vi observerer alle forespørslene som kommer inn, også de som blir avslått.

Eksempel 2

I dette eksemplet har vi bare type a og type b kunder, alle b kundene kommer før a kundene, og revenue manageren observerer ikke de forespørsler som kommer etter at prisklassen er stengt. Antall observerte forespørslene i den dyreste prisklassen blir da lik antall utleide rom i den dyreste prisklassen. Antall utleide rom i den dyreste prisklassen er lik:

$$\min \{D_a, c - \min\{D_b,(c-l)^+\}\}.$$

Den kumulative distribusjonsfunksjonen til antall observerte forespørsler i den dyreste prisklassen er da lik:

$$G(l, x) = P[\min \{D_a, c - \min\{D_b,(c-l)^+\}\} \leq x],$$

som er avhengig av l. Det vil si at den kumulative distribusjonsfunksjonen til det observerte antallet med forespørsler er avhengig av reservasjonsnivået. Antall observerte forespørsler er avhengig av reservasjonsnivået, på grunn av at vi ikke observerer de forespørslene som kommer inn etter at vi har leid ut alle rommene. Med et lavt reservasjonsnivå må vi avvise flere type a kunder enn med et høyt reservasjonsnivå, forutsatt at $D_a \geq l$ og $D_b \geq (c-l)$.

Eksempel 3

I dette eksemplet har vi med alle de tre forskjellige typene med kunder, og revenue manageren forsetter å observere kunder også etter at en har leid ut alle rommene. Antall observerte forespørsler i den dyreste prisklassen blir da lik antall forespørsler fra a kunder og antall forespørsler fra ab kunder som kommer etter at de billigste rommene ikke lenger er tilgjengelige

(altså type ab kunder som enten leier de dyreste rommene eller ankommer etter at alle rommene er leid ut). Den observerte etterspørselen blir da lik:

$$X = D_a + D_{ab} - D_{ab}(l).$$

og den kumulative distribusjonsfunksjonen til antall observerte forespørsler i den dyreste prisklassen er da lik:

$$G(l, x) = P(D_a + D_{ab} - D_{ab}(l) \leq x)$$

som er avhengig av l. Det vil si at den kumulative distribusjonsfunksjonen til det observerte antallet med forespørsler er avhengig av reservasjonsnivået. Antall observerte a kunder er avhengige av reservasjonsnivået, siden vi observerer $D_{ab}(l)$ som b kunder. Med et lavt reservasjonsnivå vil $D_{ab}(l)$ være høyere enn ved et høyt reservasjonsnivå.

Eksempel 4

I dette eksemplet har vi med alle de tre forskjellige typene med kunder og revenue manageren observerer kun de rommene som blir leid ut. Han observerer altså ikke kunder som kommer etter at alle rommene er leid ut. Den observerte etterspørselen etter de dyreste rommene X blir derfor lik antall rom som er utleid til den dyreste prisklassen. Den observerte etterspørselen etter den dyreste prisklassen X er da lik:

$$D_a(l) + \min\{D_a - D_a(l) + D_{ab} - D_{ab}(l), c, l\}$$

Vi får dermed at:

$$G(l,x) = P[D_a(l) + \min\{D_a - D_a(l) + D_{ab} - D_{ab}(l), c, l\} \leq x]$$

som er avhengig av l. Det vil si at den kumulative distribusjonsfunksjonen til det observerte antallet med forespørsler er avhengig av reservasjonsnivået. Antall observerte forespørsler er avhengig av reservasjonsnivået.

I de 3 siste eksemplene ser vi at observert etterspørsel blir påvirket av reservasjonsnivået enten på grunn av at vi har priceable etterspørsel eller på grunn av at vi ikke kan observere de forespørslene som kommer inn etter at vi har leid ut alle rommene.

Ovenfor viste jeg hvordan reservasjonsnivået kan ha innvirkning på den observerte etterspørselen etter fullpris rom. Jeg vil nå gi et eksempel på hvordan vi som en følge av dette kan få en spiral nedover av reservasjonsnivå, antall rom utleid til fullpris og inntekt. Jeg går ut fra at vi har et hotell med 10 rom og to forskjellige prisklasser (alle rommene er like). Prisen for fullpris rom er 500 kroner og prisen for tilbudsrom er 200 kroner og vi har kun priceable etterspørsel. Alle kundene ønsker å leie et rom for 200 kroner, men hvis det ikke er mulig å leie rom til tilbudsprisen, ønsker alle å leie til fullpris. Så det optimale for revenue manageren her vil være å sette reservasjonsnivået til romkapasiteten som er 10 rom. Etterspørselen er lik i hver periode og er på 8 rom (vi har en deterministisk etterspørsel). Revenue manageren går ut fra at etterspørselen er yiealdable og han lager et forecast basert på historisk observert salg. Forecast funksjonen han bruker er den empiriske distribusjonsfunksjonen av det observerte salget og han starter med å reservere 10 rom til den dyreste prisklassen.

K	Reservasjonsnivå L_{k-1}	Observert mengde X_k	Salg_Fullpris	Salg tilbudspris	Inntekt
1	10	8	8	0	4000
2	8	6	6	2	3400
3	8	6	6	2	3400
4	6	4	4	4	2800
.	.	.	.	.	.
8	6	4	4	4	2800
9	4	2	2	6	2200
.	.	.	.	.	.
20	4	2	2	6	2200
21	2	0	0	8	1600
.	.	.	.	.	.
50	2	0	0	8	1600
51	0	0	0	8	1600

Ved hjelp av denne informasjonen er det i tabell 1 regnet ut reservasjonsnivå, observert etterspørsel, salg til fullpris, salg til tilbudspris og inntekt for overnattingsdøgnene i kronologisk rekkefølge. I tabellen ser vi hvordan den feilaktige antagelsen om at vi har yieldable etterspørsel fører til en spiral nedover for inntektene, antall solgte rom til full pris og antall rom reservert til full pris. Reservasjonsnivået i tabellen er regnet ut ved hjelp av EMSR, som blir nærmere gjennomgått i kapittel 4.

Har vi en determistisk etterspørsel som er mindre enn kapasiteten, kun type ab kunder og finner forecastet ved hjelp av den empiriske distribusjonsfunksjonen vil vi alltid få en spiral nedover, lik den i tabell 1. Dette kan forklares på følgende måte:

Den observerte etterspørselen etter de dyreste rommene finner vi med følgende formel:

$$X_k = [d-(c-L_k)^+]^+$$

Hvor X_k er observert etterspørsel etter de dyreste rommene i periode k, L_k er reservasjonsnivået i periode k, c er kapasiteten og d er etterspørselen. Siden etterspørselen er mindre enn kapasiteten (d < c) har vi at observert etterspørsel er mindre eller lik reservasjonsnivået:

$$X_k=[d-(c-L_k)^+]^+ \leq [d-(c-L_k)]^+ \leq L_k$$

av dette følger det at reservasjonsnivået for periode k+1 må være mindre eller lik reservasjonsnivået for periode k. Reservasjonsnivået i periode 1 blir da lik:

$$L_1 = (\hat{H}^1)^{-1}(\gamma) = [d-(c-L_0)^+]^+ \leq L_0$$

Hvor $\hat{H}^k$ er den empiriske distribusjonsfunksjonen av den observerte etterspørselen X. Er reservasjonsnivået i den første perioden større enn null, vil reservasjonsnivået i neste periode alltid være mindre enn reservasjonsnivået i første perioden. Videre har vi at $L_{k+1} \leq L_k$ og det finnes en k* slik at alle $L_j = 0$ og alle $X_j=0$ for $j \geq k^*$. Vi får en spiral nedover som vil ende med at vi ikke reserverer noen rom til den dyreste prisklassen. Spiralen nedover kan ses i tabell 1 (Cooper, Homem-de-Mello & Kleywegt, 2006).

Forecast

For at hotellet skal kunne sette gode reservasjonsnivåer og bookinglimiter trengs et godt forecast. Forecastet må gi svar på hvor mange som er interessert i å leie rom i de forskjellige prisklassene. Å ha et godt forecast er viktig, for har vi et dårlig forecast vil dette føre til tap av inntekter. Tap av inntekter med et dårlig forecast skyldes at vi setter feil bookinglimit og reservasjonsnivå. Et dårlig forecast kan også føre til spiral down effekten.

Det er imidlertid veldig vanskelig å få til et nøyaktig forecast på grunn av at det er så mange faktorer som spiller inn på etterspørselen. Nedenfor følger noen faktorer som spiller inn på etterspørselen, disse er hentet fra Mcgill og van Ryzin (1999):

Sesongsvingninger: Etterspørselen vil variere avhengig av om det er sommer eller vinter. For eksempel i Bergen, som er en turistby, vil etterspørselen etter hotellrom være større om sommeren enn om vinteren.

Ukedag: Det er større belegg av forretningsreisende i ukedagene enn i helgene.

Spesielle begivenheter: Under festspillene i Bergen vil det for eksempel være større etterspørsel etter hotellrom enn ellers.

Sensitivitet i forhold til pris: Økning eller reduksjon i pris vil føre til henholdsvis etterspørselsøkning eller etterspørselsreduksjon, men forskjellige kundegrupper vil ha ulik pris elastisitet.

Etterspørselsavhengighet mellom prisklassene: Kunder som leier rom i den billigste prisklassen hadde kanskje leiet rom i en dyrere prisklasse hvis ikke den billigste prisklassen hadde vært tilgjengelig. Kunder som leier rom i den dyreste prisklassen hadde kanskje leiet rom i en billigere prisklasse hadde dette vært mulig.

Avbestillinger: En må ta hensyn til at noen kunder reserverer rom for siden å avbestille dem.

No-shows: Noen bestiller rom og bestemmer seg for ikke å bruke dem uten å avbestille.

Sensurering av historisk etterspørselsdata: De forespørslene som kommer etter en prisklasse som ikke lenger er til salgs vil ikke bli registrert.

Fulle hoteller: Kan føre til at kunder velger å overnatte på andre datoer enn det som opprinnelig var planlagt.

Forecasting metoder

Jeg vil nå ta for meg 2 forecasting metoder som begge blir brukt i revenue management i dag. De 2 metodene som blir gjennomgått er exponential smoothing og moving average. Begge disse metodene gir et forecast på etterspørselen etter rom i de forskjellige prisklassene, ved bruk av historisk booking data.

Exponential smoothing

Exponential smoothing er en forecasting metode som bruker siste forecast og den siste observasjonen til å lage et nytt forecast. Modellen er enkel å bruke og krever kun lagring av små mengder med data. Siden modellen kun bruker den siste observasjonen og det siste forecastet, er dette alt vi trenger å lagre for å bruke modellen. For å kunne estimere et nytt forecast, må vi bruke en smoothing parameter ά. Denne paremeteren bestemmer hvor mye vekt som skal legges på den siste observasjonen i forhold til de andre observasjonene. Modellen kan se ut som følgende:

$Y_{t+1} = ά * X_t + (1-ά)*Y_t \qquad ά(E)[0,1]$

Hvor X_t står for den siste observasjonen og Y_t er det siste forecastet. Valget av parameteren ά vil ha stor betydning for hvor rask responstiden til modellen vil være. Har parameteren lav verdi vil modellen respondere sakte til endringer i data, noe som resulterer i et forholdsvis stabilt forecast. For større verdier av parameteren vil responstiden være raskere. Dessverre kan endringen i data reflektere både en ny trend eller bare tilfeldige variasjoner. Det siste kan føre til et uønsket, ustabilt forecast. De konkurrerende behovene for et forecast som er stabilt og et som tar hensyn til forandring av gjennomsnitt, må en ta hensyn til når en velger parameter.

Er ά<1 brukes alle tidligere perioder av observert etterspørsel og er ά=1 brukes kun den siste observasjonen. Dette kan en se ved å skrive om ligningen:

$Y_t=\alpha X_{t-1} + (1- \alpha)\alpha X_{t-2} +(1 -\alpha)^2 Y_{t-2}$

$Y_t=\sum^{\infty}_{i=0}(1-\alpha)^i \alpha X_{t-i-1}=\sum^{\infty}_{i=0}\alpha_i X_{t-i-1}$

Hvor vi ser at hver eneste av de tidligere observasjonene er med på å produsere det nyeste forecastet, men vekten gitt til hver observasjon blir redusert eksponentialt (Zeni, 2001).

Moving average

Denne modellen lager et forecast for den fremtidige etterspørselen ved å finne gjennomsnittet av de n siste observasjonene. Modellen blir kalt moving average på grunn av at gjennomsnittet endrer seg med tiden, ved at en dropper den eldste observasjonen og tar inn den siste observasjonen. Forecastet for tidsperiode t+1 blir kalkulert med følgende formel:

$$\text{Forcast}_{t+1} = 1/n \sum_{k=t}^{t-n+1} \text{observasjon}_k$$

Moving average modellen er lett å forstå og bruke. For å bruke modellen må vi lagre de n siste observasjonene, vi må derfor lagre mer data enn ved exponetial smoothing. Exponential smoothing og moving average kan brukes til å gi et forecast på data som varierer rundt et konstant gjennomsnitt. Men hvis etterspørselen har en positiv eller negativ trend vil disse metodene alltid gi et forecast som ligger etter trenden og vi vil få et feilaktig forecast (Zeni, 2001).

Trender

Holt har laget en metode for å ta hensyn til trender i forecastet. I Holts metode regner vi ut to verdier, den ene gir det forventede nivået (R) til etterspørselen og den andre gir et estimat over

trenden (G). Disse to summeres og vi får et forecast for neste periode. Forecastet for neste periode kan da se slik ut:

$Y_t = R_{t-1} + G_{t-1}$

Ideen i Holts metode er å kombinere vårt beste estimat for trenden med vårt beste estimat over nivået (i periode t-1) for å finne et forecast. Trendestimatet representerer den endringen i etterspørselen vi forventer fra denne perioden til den neste. For å lage et forecast må vi starte med å lage et estimat på nivået, dette gjøres ved å kombinere denne periodens etterspørsel X_t med siste periodes forecast for etterspørselen:

$R_t = \alpha * X_t + (1 - \alpha) * Y_t = \alpha * X_t + (1-\alpha) * (R_{t-1} + G_{t-1})$

Hvor $\alpha \in [0, 1]$ er en konstant som settes inn for å bestemme vekten på denne periodens etterspørsel X_t i forhold til periodens forecast Y_t. Etter å ha estimert nivået kan en lage et estimat på trenden. Dette gjøres ved å kombinere et nytt trendestimat med det gamle trendestimatet:

$G_t = \beta(R_t - R_{t-1}) + (1- \beta)G_{t-1}$

Hvor $\beta \in [0, 1]$ er en konstant som settes inn for å bestemme vekten til det nye trendestimatet i forhold til siste periodes trendestimat G_{t-1}. Det nye trendestimatet er i ligningen $R_t - R_{t-1}$, som er forskjellen på det siste nivået og det nye nivået (Zeni, 2001).

Forecast med sesongsvingninger

Hoteller har ofte etterspørsel som er sesongbetont. Med begrepet sesong menes her tendenser i etterspørselen som gjentar seg selv, innenfor faste tidsintervaller. En bruker termen sesong til å representere perioden med tid før etterspørselen begynner å repetere seg selv igjen, q er sesongens lengde i perioder.

Vi har flere forskjellige typer av sesongmessig etterspørsel innenfor hotellbransjen. Hvis en for eksempel ser på etterspørselen uke etter uke, vil en se at etterspørselen vil variere fra ukedag til ukedag. Det kan for eksempel være lav etterspørsel fra forretningsreisende på å leie rom i helgene, mens det er stor etterspørsel midt i ukene. I dette tilfellet vil sesongen bestå av 7 perioder, en periode er en dag. Etterspørselen kan også svinge med årstiden, for eksempel med stor etterspørsel om sommeren, lav om vinteren mens vår og høst har middels etterspørsel. Da vil sesongen bestå av 4 perioder.

Hvis den prosentmessige forskjellen mellom tidsperiodene i sesongen ikke endrer seg med tiden, vil enn kunne bruke konstante multiplikative justeringsfaktorer for hver periode. En kan for eksempel bruke en sesongjustert tidsserie, $\overline{X}_i$ i=1,2,…,t-1 definert som følger:

$$\overline{X}_i = X_i/S_j \qquad j = i \bmod q$$

Hvor X_i er den observerte etterspørselen i periode i, og S_j er sesongjusteringsfaktoren til perioden j i sesongen. For å finne de sesongmessige justeringsfaktorene må en først kalkulere et sesongjustert nivå $\overline{X}_t$ for hver periode. Hvis en ikke har noen trend, er det sesongjusterte nivået gjennomsnittsverdien til alle periodene. Hvis etterspørselen inneholder en trend, kan en bruke en enkel lineær regresjons modell til å lage et estimat på $\overline{X}_t$. Hver justeringsfaktor S_j kan da bli estimert ved å ta snittet over like perioder for hver sesong hvor data er tilgjengelig:

$$S_j = \text{gjennomsnittet}\left(\frac{X_j}{\overline{X}_j}, \frac{X_{j+q}}{\overline{X}_{j+q}}, \frac{X_{j+2q}}{\overline{X}_{j+2q}}, \ldots\right)$$

Sesongjusteringsfaktorene bør ha gjennomsnittsverdi på 1, vi har derfor at

$$q = \sum_{j=1}^{q} S_j$$

Teorien i dette stykket er hentet fra Zeni, 2001.

Winters Metode

Winter har laget en metode, hvor han bruker trippel eksponentiell smoothing tilnærming for å kunne ta hensyn til både trenden og sesongsvingninger. I Winters metode lages et estimat over det sesongjusterte nivået $\overline{R}_t$, et estimat over trenden $\overline{G}_t$, og et estimat over den multiplikative sesongfaktoren S_t. En sesong deles opp i q perioder. For notasjonsgrunner, vil jeg gi en sesongjusteringsfaktor S_t for hver periode t, istedetfor bare de første q sesongperiodene. Gitt disse tre verdiene, vil vårt forecast for neste periode være:

$$Y_t = (\overline{R}_{t-1} + \overline{G}_{t-1})S_{t-q}$$

I ligningen ganges det sesongjusterte forecastet med sesongjusteringsfaktoren for å få forecastet for perioden. Det er en sesong justeringsfaktor for hver periode i sesongen. Nedenfor går jeg kort gjennom hvordan vi kan oppdatere nivået, trend og sesong justeringsfaktorene (en for hver tidsperiode).

Først oppdaterer vi det sesongjusterte nivået som følger:

$$\overline{R}_t = \alpha \frac{X_t}{S_{t-q}} + (1-\alpha)(\overline{R}_{t-1} + \overline{G}_{t-1})$$

Dette er likt som i Holts metode, bortsett fra at vi har sesongjustert etterspørselsobservasjonen i den første termen og brukt det sesongjusterte forecast i den andre termen. Etter dette oppdaterer vi trenden:

$$\overline{G}_t = \beta(\overline{R}_t - \overline{R}_{t-1}) + (1-\beta)\overline{G}_{t-1}$$

Vi kombinerer et nytt sesongjustert trendestimat med vårt tidligere trendestimat. Sesongjusteringsfaktoren er en kombinasjon av den siste observerte etterspørsel X_t, delt på det sesongjusterte nivået $\overline{R}_t$ og det tidligere sesongjusteringsfaktor estimatet (S_{t-q}) for denne tidsperioden:

$$S_t = \gamma \frac{X_t}{\overline{R}_t} + (1-\gamma)S_{t-q}$$

konstanten y E [0,1] bestemmer hvilken vekt enn legger på den siste observerte sesongvariasjonen i forhold til det tidligere sesongjusteringsestimatet (Zeni, 2001).

Sensurert data

Med sensurert data menes her etterspørsel etter hotellrom som er ukjent for hotellet. Etterspørsel som vil være ukjent for hotellet er etterspørsel etter en prisklasse som kommer inn etter at denne prisklassen ikke lenger er åpen for utleie. Forutsatt at hotellet ikke fortsetter å registrere antall forespørsler som kommer inn etter at de har sluttet å leie ut rom i prisklassen. Et eksempel på sensurert data er når hotellet velger å ikke leie ut flere rom til prisklasse 2, 14 dager før siste salgstidspunkt. Da er forespørslene som kommer inn for prisklasse 2 i disse 14 dagene ukjent for hotellet. En vet ikke hvor stor etterspørselen etter å leie rom i prisklasse 2 har vært i disse 14 dagene, ettersom en ikke har registrert dette i denne perioden. En har da sensurert data i en 14 dagers periode. Hvis hotellet holder muligheten til å leie rom i en prisklasse åpen helt til siste salgstidspunkt, vil hotellet vite den totale etterspørselen etter rom i denne prisklassen. Etterspørselen til prisklassen er da lik antall utleide rom i prisklassen. Når vi ikke har skjult etterspørsel kaller vi det for usensurert data.

Jeg vil nedenfor gå gjennom ulike metoder som hotellet kan bruke for å ta hensyn til sensurert data. I noen av disse metodene gjør jeg bruk av observasjonspunkt. Observasjonspunkt er et bestemt tidspunkt t mellom [0, N], hvor en foretar en observasjon av antall forespørsler som er kommet til nå i prisklassen. Er bookinglimiten til prisklasse i, mindre enn etterspørselen etter

prisklasse i ved observasjonspunkt r, har vi sensurert data ved observasjonspunkt r. Vi må da estimere etterspørselen fra observasjonspunkt r-1 og frem til observasjonspunktet r.

Det letteste alternativet for hotellet er å ignorere at en har sensurert data. Velger hotellet å ignorere at en har sensurerte data vil dette føre til negativt bias. Dette på grunn av at en da vil mangle de forespørslene som kommer inn etter at bookingklassen er stengt. Et annet alternativ for hotellet, er å forkaste de sensurerte dataene. Dette alternativet vil begrense den gjenværende sample-størrelsen. Den begrensede sample-størrelsen vil sannsynligvis bestå av flest tilfeller med lav etterspørsel, på grunn av at sannsynligheten er størst for at vi forkaster tilfeller med høy etterspørsel. Dette skjer fordi sannsynligheten for å nå bookinglimiten er størst når vi har høy etterspørsel. Det vil føre til et negativt bias, siden vi forkaster tilfellene med høy etterspørsel og beholder de med lav etterspørsel. Det er også mulig å få en positiv bias. Dette skjer der en forkaster flest av tilfellene med lav etterspørsel. Dette scenariet er veldig usannsynlig, siden det ligger i sakens natur at det er lettere å nå bookinglimiten med høy etterspørsel.

En sensitivitetsanalyse utført av Weatherford (1997) har demonstrert kostnaden av å bruke et negativt bias forecast i et Yieald management system. Når forcastet er 12,5 % lavere enn faktisk etterspørsel, kan inntektene reduseres med så mye som 0,7 til 1.2 prosent. Med forecast som er 25 % lavere enn faktisk etterspørsel, kan inntekten reduseres så mye ned som 2 til 3 prosent. Metodene ovenfor tar egentlig ikke hensyn til problemet, men håper bare at problemet ikke er for stort. Hvis sensurerte data er begrenset til en veldig liten del av dataene, er kanskje ikke kostnaden med å innføre en mer avansert teknikk verdt implementeringen og vedlikeholdskostnaden til modellen.

Det beste ville ha vært å ikke ha sensurert data. Sensurert data ville en kunne ha unngått hvis hotellet hadde registrere alle forespørslene, også de som kom inn etter at bookinglimiten var nådd. Noe som kunne vært gjort ved at hotellet fortsatte registreringen av forespørslene også etter at en prisklasse var stengt. Dette kan være vanskelig av flere grunner. En grunn er at alle reservasjonene ikke nødvendigvis går gjennom hotellet. Kunder kan for eksempel reservere rom gjennom reisebyrå, istedetfor direkte fra hotellet. Hotellet kunne likevel registrert antall forespørsler som kom gjennom sine egne bookingkontorer. Da ville de ha mye bedre kontroll

over datainnsamlingsprosessen og kunne ha skalert opp tallene. Dette ville imidlertid ha gitt reservasjonsagentene ekstra arbeidsoppgaver som ville ha krevd flere ressurser. Nytten av dataene ville kanskje ikke ha oversteget arbeidet med å samle dem inn. Det er også en problemstilling hvordan hotellet i praksis skulle klare å registrere antall forespørsler som kommer etter at bookingklassen er stengt.

Mean imputation metoden er en blanding mellom det å fjerne sensurerte observasjoner og det å ignorere dem. Metoden sammenligner antall reservasjoner på et bestemt tidspunkt (observasjonspunkt) med bookinglimitene. Er antall reservasjoner ved dette tidspunktet likt som bookinglimiten, har en sensurert data. Ved mindre reservasjoner enn bookinglimiten, har en ikke sensurert data. De observasjonene som ikke har sensurert data blir brukt som de er. De observasjonene som har sensurert data blir sammenlignet med gjennomsnittet til de observasjonene som ikke er sensurert. Er antall reservasjoner mindre enn gjennomsnittet blir disse erstattet av gjennomsnittet. Er det flere reservasjoner enn gjennomsnittet blir de brukt som de er. Etterspørselsestimatet på hvert punkt blir da som følgende:

$$AB(r) = \frac{\sum_{k=1}^{n} CB(r,k)I(r,k)}{\sum_{k=1}^{n} I(r,k)}$$

hvor CB(r, k) er antall bestillinger for prisklasse k på tidspunkt r og I(r, k) er en indikator på om prisklasse k er åpen eller stengt på tidspunkt r. Er prisklassen stengt er I(r, k) lik 0 og er den åpen er I(r, k) lik 1. Fordelen med denne metoden er at den er lett å implementere og kan på en enkel måte avdekke noe av dataene. Ulempen er at den avdekkede observasjonen høyst sannsynlig vil underestimere den virkelige etterspørselen. Har en for eksempel en sensurert observasjon som er under gjennomsnittet til de usensurerte observasjonene, vil denne sensurerte observasjonen bli erstattet av gjennomsnittet til de usensurerte observasjonene. Problemet er at dette gjennomsnittet også vil være sensurert. Gjennomsnittet består av de data som ikke var sensurert, men disse var usensurert fordi etterspørselen var lav. Hvis etterspørselen var høy, ville observasjonen høyst sannsynlig vært sensurert og da hadde de ikke blitt inkludert i gjennomsnittet. Ved å bruke

gjennomsnittet som input vil en bevege seg nærmere den virkelige etterspørselen, men en vil fortsatt høyst sannsynlig ha en underestimering av etterspørselen. Istedetfor å bruke gjennomsnittet i denne metoden, kan en for eksempel bruke medianen.

Multiplikativ booking profil metoden finner den gjennomsnittlige prosentmessige økningen til de usensurerte dataene mellom alle observasjonspunktene, og bruker så den gjennomsnittlige prosentmessige økningen til å estimere den virkelige etterspørselen til de sensurerte dataene. I denne modellen antar en at den underliggende formen til bookingprofilen er lik, uavhengig av nivået på etterspørselen. Det betyr at for en gitt prisklasse i, vil formen på bookingprofilen være lik uansett antall forespørsler. Med andre ord, den prosentmessige økningen mellom observasjonspunktene er konstant for en gruppe overnattingsnetter på samme hotell, med likt tidspunkt i sesongen, lik ukedag osv. Hvis dette stemmer, kan bookingprofilen bli nøyaktig estimert ved å bruke gjennomsnittlig økning i antall forespørsler på observasjonspunktet for en gruppe tilsvarende overnattingsnetter. De sensurerte observasjonene kan da bli avdekket ved å multiplisere det usensurerte observasjonstidspunktet med den prosentmessige økningen.

Steg for steg virker den multiplikative booking profil metoden på følgende måte, der en starter med det første observasjonstidspunktet:

Steg 1: Finn det gjennomsnittlige antallet med forespørsler for prisklasse i, for observasjonspunktene r og r - 1, for de n siste usensurerte observasjonene i de sammenlignbare overnattingsdøgnene.

$$AB(r) = \frac{\sum_{k=1}^{n} CB(r,k)I(r,k)}{\sum_{k-1}^{n} I(r,k)}$$

Steg 2: Finn den gjennomsnittlige økningen i antall forespørsler for prisklasse i mellom observasjonspunktene (r, r-1):

$$PI(r, r-1) = AB(r) / AB(r-1)$$

Steg 3: Estimer antall forespørsler for de sensurerte dataene for observasjonspunkt r og sørg for at antall estimerte forespørsler er lik eller større enn de observerte forespørslene.

$$UD(r,k) = \begin{cases} CB(r,k) & \text{vis usensurert data} \\ Max[CB(r-1,k)PI(r,r-1)CB(r,x)] & \text{vis sensurert data} \end{cases}$$

Steg 4: Repetèr steg en til steg 3 for alle gjenværende observasjonspunkt..

Fordelen med denne metoden er at den er lett å implementere og det er ikke noen begrensninger i hvor mange forventede forespørsler en kan få i den estimerte etterspørselen. Dette til forskjell fra mean imputation metoden som ikke kan gi et estimat som er større enn kapasiteten på hotellet. Ulempen med booking profil metoden et at den ignorere for mye data (Zeni, 2001).

Expectation-maximization (EM)

Expectation-maximization algoritmen er en algoritme for å finne det mest sannsynlige estimatet på forventningen og variansen til et ufullstendig dataproblem, men den kan også bli brukt til å estimere verdiene til manglende data. Algoritmen kan brukes i situasjoner hvor en har sensurerte observasjoner, manglende data og avkortet distribusjon. Tanken bak EM algoritmen er å ta det ufullstendige dataproblemet og sammenligne det med et fullstendig dataproblem, for å kunne finne et bedre estimat på forventning og varians. EM algoritmen består av to deler: en forventningsdel og en maksimeringsdel. Forventningsdelen løses først. Den lager data til det

fullstendige dataproblemet ved å regne ut den betingede forventningen til de uobserverte dataene, gitt de observerte forespørslene og nåværende parameterverdier. Maksimeringsdelen blir så brukt på det fullstendige datasettet, for å finne den mest sannsynlige forventningen og variansen. Dette blir så repetert til en har konvergens.

For å estimere etterspørselen til hotellet undersøker en først om en har sensurert eller usensurert data ved hvert observasjonspunkt. Hvis en har usensurert data, bruker en observasjonen som den er til å representere den virkelige inkrementelle etterspørselen mellom hvert observasjonspunkt. Hvis indikatoren er sensurert bruker en EM algoritmen til å finne den inkrementelle etterspørselen mellom observasjonspunktene.

For å finne de usensurerte observasjonene bruker vi EM algoritmen på følgende måte:

1 Først undersøker vi om det er sensurert eller usensurert etterspørsel mellom observasjonspunkt r og r + 1, har vi usensurert etterspørsel er den observerte etterspørselen UD(r, k) lik den virkelige etterspørselen.

   $$UD(r, k) = IB(r, k) + UD(r + 1, k)$$

   Hvor IB(r, k) er den inkrementelle etterspørselen for observasjonspunkt r, som er forskjellen i antall forespørsler mellom observasjonspunkt r og observasjonspunkt r + 1

   Har en sensurert etterspørsel bruker en EM algoritmen til å estimere den inkrementelle etterspørselen mellom de aktuelle observasjonspunktene. Hvor ID(r, k) er den estimerte inkrementelle etterspørselen for observasjonspunkt r, som er den estimerte forskjellen i antall

forespørsler mellom observasjonspunkt r og observasjonspunkt r + 1. Algoritmen estimerer så den skjulte etterspørsel ved observasjonspunkt r som følger:

$$UD(r,k)=\begin{cases} IB(r,k)+UD(r+1,k) & \text{vis usensurert data} \\ ID(r,k)+UD(r+1,k) & \text{vis sensurert data} \end{cases}$$

hvor

$$ID(r, k) = E[x \mid x > IB(r, k)]$$

2 Deretter utarbeides et første estimat på forventningen og variansen. Det første estimatet på forventningen og variansen til hvert observasjonspunkt blir regnet ut fra den usensurerte historien til n lignende overnattingsnetter:

$$\mu(r)^{(0)}=\sum^{n}_{k=1} IB(r, k)/n$$
$$\sigma^2(r)^{(0)}=\sum^{n}_{k=1}(IB(r, k)-\mu(r)^{(0)})^2/(n-1)$$

hvis også alle de tidligere observasjonene er sensurerte, vil ikke ligningen over produsere gjennomsnitt og varians. Derfor, hvis alle data er sensurert, er initial estimatet til den avdekte etterspørselen lik de observerte inkrementelle bookingene:

$$UD(r, k) = IB(r, k) + UD(r+1, k)$$

3 E delen erstatter de sensurerte observasjonene med den forventede verdien kalkulert med ligning:

$$E[x|x > IB(r,k)] = \frac{\int_{IB(r,k)}^{\infty} xf(x)dx}{\int_{IB(r,k)}^{\infty} f(x)dx}$$

4 Ved hver gjentagelse t regnes μ og varians ut på nytt, med de nye dataene:

$\mu(r)^{(t)} = 2(\sum_{k=1}^{\infty} IB(r, k) + ID(r, k)^{(t-1)})/n$

$\sigma^2(r)^{(t)} = \sum_{k=1}^{n}(IB(r, k) - \mu(r)^{(t)})^2 + \sum_{k=1}^{n} (ID(r, k)^{(t-1)} - \mu(r)^{(t)})^2$

5 Steg 3 og 4 repeteres inntil konvergens. Konvergens er oppnådd når forskjellen mellom gjennomsnittene er mindre eller lik Q:

$Q > |\mu(r)^{(t)} - \mu(r)^{(t-1)}|$

hvor Q er et lite tall større enn null.

Teorien i dette stykket er hentet fra Zeni, 2001.

Kapittel 4

I dette kapitlet blir teorien til de ulike løsningsmetodene for romallokeringen på et hotell beskrevet. Kapitlet starter med å gå gjennom de statiske metodene EMSRA og EMSRB hvor det kun er mulig å leie ett rom for en natt. Deretter tar kapitlet for seg dynamisk programmering og avsluttes med en gjennomgang av Popescu og Bertsimas sin teori om Certainty equivalent control.

EMSR (Excpected marginal substitusjons rate)

EMSR bruker en for å bestemme hvor mange rom som skal reserveres til de dyreste prisklassene og hvor mange rom som kan leies ut til de biligste prisklassene. For å forstå tanken bak EMSR ser jeg først på et hotell med bare to prisklasser (fullpris og tilbudspris). Dette hotellet må bestemme seg for hvor mange rom det ønsker å reservere til fullpris kunder. Reserveres for mange rom, risikerer en at noen av rommene vil stå tomme. Reserveres for få rom vil en risikere å måtte avvise kunder som er villige til å betale fullpris. En må derfor finne en balansegang mellom det å tape penger på grunn av at rommene står tomme, og det å tape penger pågrunn av at en har leiet ut et rom til tilbudspris, som kunne ha vært leiet ut til fullpris. Jeg lar r_h stå for antall rom reservert til kunder som betaler fullpris. De rommene som ikke blir reservert til fullpris kunder kan leies ut til tilbudspris. Det antall rom som kan leies ut til tilbudspris kalles bookinglimit. Bookinglimit'en vil en finne på følgende måte:

$$\text{Bookinglimit} = \text{totalt antall hotellrom} - r_h$$

Bookinglimit'en sier hvor mange rom som kan leies ut til tilbudspris.

For at hotellet skal finne den optimale bookinglimit'en og det optimale reservasjonsnivået, må en ha sannsynligheten for å ikke få leiet ut alle rommene reservert til fullpris kunder for alle reservasjonsnivå (r_h=1,2….,n). Har en valgt å holde tilbake r_h rom til fullpris, vil sannsynligheten for at alle som etterspør et rom til fullpris får dette være lik:

$$P[X \leq r_h] = P(0) + \ldots\ldots + P(r_h)$$

Hvor P(i) er sannsynligheten for å få en etterspørsel på akkurat i rom. Sannsynligheten for ikke å ha rom til alle kundene som etterspør rom til fullpris vil da være lik:

$$P[X > r_h] = 1 - P[X \leq r_h]$$

EMSR står for den forventede marginale romfortjenesten av å øke antall reserverte rom til

fullpris med èn. EMSR finner vi ved å ta prisen på fullpris rommene og gange den med

sannsynligheten for å få leid ut mer enn r_h rom $P[X> r_h]$:

$$EMSR(r_h)=høypris \cdot P[X> r_h]$$

For å oppnå den optimale romallokeringen, blir antall rom reservert til fullpris kunder økt med

èn, så lenge som EMSR er større eller lik prisen på et tilbudsrom. r_h økes derfor med èn så lenge som:

$$EMSR(r_h) \geq lavpris \text{ eller } høypris \cdot P[X> r_h] \geq lavpris$$

Verdien som da fåes på r_h er det antallet med rom som reservers til fullpris, og booking limit blir da lik antall hotellrom – r_h (Belobaba, 1989)**.**

Flere prisklasser

Hotellene har ofte mer enn to prisklasser. Derfor bør løsningsmetodene også kunne ta hensyn til at det er mer enn to prisklasser. Beloba har to modeller som løser dette: EMSRa og EMSRb, hvor EMSRa kom først, og EMSRb er en forbedring av EMSRa metoden. Jeg vil først gjøre rede for EMSRa, for deretter å gjøre rede for EMSRb.

EMSRa

EMSRa brukt på flere prisklasser enn to, finner rom allokeringen på samme måte som beskrevet ovenfor hvor det kun var to prisklasser. Metoden starter med å se på hvor mange rom som skal reserveres for den dyrest prisklassen, og som ikke kan leies ut til prisklasse 2 eller billigere prisklasser. Etter å ha reservert rom for prisklasse 1, finner vi antall rom som skal reserveres for prisklasse 2 og ikke kan leies ut til prisklasse 3 eller billigere prisklasser. Slik fortsetter en helt til

en har reservert rom til alle prisklassene som har en eller flere billigere prisklasser. Dette kan vises med følgende formel hvor r_{ij} står for antall rom beskyttet for klasse i fra klasse j. Antall rom

som blir reservert til klasse i fra klasse j blir bestemt med å øke r_{ij} med en så lenge som

$$I_i \cdot P(X_i < r_{ij}) \geq I_j \qquad i < j, \quad j = 1,\ldots, k,$$

Hvor I_i er prisen (inntekten) for rom i og k er antall prisklasser. Når antall rom reservert for hver enkelt prisklasse er bestemt, gjenstår det å finne bookinglimit for hver klasse. Bookinglimit for pris klasse j, BL_J, er maksimalt antall rom tilgjenglig for pris klassene j, j+1,….,k. Bookinglimit for klasse j er gitt ved

$$BL_j = \text{maks}[\ 0,\ C - \sum_{i<j} r_{ij}\ ], \quad j = 1,\ldots\ldots,k.$$

Hvor C er antall rom på hotellet. En ser at bookinglimit`en blir kalkulert ved å trekke fra summen av beskyttede rom for klasse j og alle andre dyrere prisklasser, siden en har ”nested bookinglimit”. Ved nesting vil fullprisklassene ta rom på bekostning av tilbudsprisklassene, hvis de trenger det.

Ettersom prisklassene blir sammenlignet parvis, vil metoden sannsynligvis ikke gi den optimale romallokeringen når en har mer enn to prisklasser. Beloba har tatt hensyn til dette i EMSRb.

EMSR modellene (a og b) er statiske, men problemet som skal løses er dynamisk. For å løse det dynamiske problemet, blir den statiske modellen løst flere ganger med ny input data (etterhvert som tiden går). Når modellen blir ”løst” for 2. gang, vil for eksempel bookingen så langt være kjent. Romallokeringen kan dermed oppdateres på følgende måte:

$$\sum_{i<j} r_{ij} = \sum_{i<j} r_{ij}(t) + \sum_{i<j} b_{ti}$$

hvor $r_{ij}(t)$ er romallokeringen kalkulert på tidspunkt t ut fra etterspørselsforventningen på tidspunkt t og frem til siste bestillingstidspunkt og $\sum_{i<j} b_{ti}$ er antall aksepterte forespørsler for

prisklasse i og alle dyrere prisklasser. Dette gir:

$$BL_j = C - \sum_{i<j} r_{ij}(t) - \sum_{i<j} b_{ti}$$

Bookinglimiten til prisklasse j er altså antall rom på hotellet, minus antall rom en ønsker å reservere til dyrere prisklasser fra tidspunkt t frem til siste salgstidspunkt, minus antall rom en allerede har leiet ut til dyrere prisklasser. Bookinglimiten for prisklasse j er nødt til å være større eller lik null, og den kan ikke være mindre enn summen av de rommene en har akseptert for prisklasse j og alle billigere prisklasser. Dette gir:

$$BL_j(t) = \text{maks}[C - \sum_{i<j} r_{ij}(t) - \sum_{i<j} b_{it}, \sum_{l\geq j} b_{lt}, 0] \qquad \textbf{(4.1)}$$

Hvor $\sum_{l\geq j} b_{lt}$ er summen av de forespørslene en allerede har akseptert for pris klasse j og alle billigere prisklasser (Belobaba, 1989).

EMSRb

I EMSRb løsningsmetoden finner vi et felles reservasjonsnivå π_i, for alle prisklassene som har et dyrere prisnivå enn prisklasse (i + 1) som er den prisklassen vi skal finne bookinglimiten til. Det felles reservasjonsnivået blir beregnet ut fra gjennomsnittsprisen $I_{1,i}$ til de forventede forespørslene i prisklasse i og alle dyrere prisklasser. Vi starter med å beregne π_1, π_2 osv. Reservasjonsnivået for pris klasse 1 bestemmer en ved å finne den største verdien av π_1 som oppfyller følgende ligning:

$I_{1,1} \cdot P(x_1 < \pi_1) \geq I_2$

hvor

$I_{1,1} = I_1$

Vi bruker så π_1 til å finne bookinglimiten for pris klasse 2, ved bruk av formelen:

$BL_2 = C - \pi_1$

For å finne bookinglimiten for prisklasse 3, brukes det felles reservasjonsnivået for pris klasse 1 og 2. Dette finner vi ved å bruke en kombinasjon av etterspørsel og prisnivå for disse to klassene:

$E(r_{1,2}) = E(r_1) + E(r_2)$
$I_{1,2} = (I_1 \cdot E(r_1) + I_2 \cdot E(r_2)) / E(r_{1,2})$
$\Phi_{1,2}(\pi_2) = P(x_1 + x_2 \geq \pi_2)$
$EMSRb_{1,2}(\pi_2) = I_{1,2} \cdot \Phi_{1,2}(\pi_2)$

hvor r_i er den tilfeldige etterspørselen for prisklasse i, $\Phi_{1,2}(\pi_2)$ er sannsynligheten for at etterspørselen etter prisklasse 1 og 2 er større eller lik π_2 og $E(r_i)$ er forventet verdi av r_i. Optimalt reservasjonsnivå er den største verdien av π_2 som tilfredstiller:

$EMSRb_{1,2}(\pi_2) \geq I_3$.

Vi kan nå finne bookinglimiten for prisklasse 3 ved å bruke det felles reservasjonsnivået for prisklasse 1 og 2:

$BL_3 = C - \pi_2$.

Bookinglimiten for prisklasse i+1 finner vi ved å bruke det felles reservasjonsnivået for prisklasse i og alle dyrere prisklasser. Dette gjør vi ved å kalkulere som følgende:

$$E(r_{1,i}) = \sum_{n=1}^{i} E(r_n)$$

$$I_{1,i} = \sum_{n=1}^{i} (I_n \cdot E(r_n)) / E(r_{1,i})$$

$$\Phi_{1,i}(\pi_i) = P(\sum_{n=1}^{i} x_n \geq \pi_i)$$

$$EMSRb_{1,i}(\pi_i) = I_{1,i} \cdot \Phi_{1,i}(\pi_i)$$

Vi må så finne den største verdien av π_i som tilfredstiller ligningen:

$$EMSRb_{1,i}(\pi_i) \geq I_{i+1}$$

Bookinglimiten for prisklasse i +1 finner vi da ved hjelp av følgende formel:

$$BL_{i+1} = C - \pi_i$$

EMSRb metoden kan brukes på et dynamisk problem på samme måte som EMSRa metoden, hvor bookinglimiten oppdateres på samme måte. Dette gir:

$$BL_{i+1}(t) = \text{maks}(C - \pi_i - \sum_{j<i+1} b_{tj}, \sum_{l<i+1} b_{tl}, 0)$$

Hvor b_{tn} er antall aksepterte forespørsler for prisklasse n på tidspunkt t. EMSRb metoden gir bedre resultater enn EMSRa metoden. Dette fordi den tar hensyn til alle dyrere prisklasser ved fastsettelse av reservasjonsnivået og ikke bare en prisklasse slik som i EMSRa (Belobaba, 1992).

Dynamisk programmeringsmodell

Problemet som skal løses er et dynamisk problem og som nevnt tidligere i oppgaven er EMSR modellene statiske modeller. En svakhet med de statiske modellene er at de har problemer med å ta hensyn til tidsavhengige variasjoner i etterspørselen, ettersom det blir antatt at etterspørsel fra nåtid til siste salgstidspunkt kan beskrives av en enkel tilfeldig variabel. Den statiske EMSR modellen ble likevel brukt i denne oppgaven, ettersom det kan ta veldig lang tid å løse en dynamisk programmeringsmodell. For å ta hensyn til variasjonen i etterspørslen har jeg også tatt med en dynamisk programmeringsmodell for hotellet. Jeg har blant annet sett på Lee og Hersh sin artikkel hvor de har satt opp en dynamisk programmeringsmodell. Den dynamiske programmeringsmodellen vil først beskrives på en intuitiv måte for så å vises matematisk. I modellen har jeg brukt følgende forenklede antagelser:

1. Etterspørselen til de forskjellige prisklassene er uavhengig av hverandre, det vil si at etterspørselen etter en prisklasse påvirker ikke etterspørselen etter en annen prisklasse.

2. Etterspørselen er modellert som en stokastisk prosess; etterspørselsdistribusjonen er antatt og vil være kjent.

3. Etterspørselssannsynligheten vil variere med tid.

4. Det vil ikke være mer enn en etterspørsel etter rom i løpet av en beslutningsperiode.

5. En må enten akseptere eller avslå forespørselen hver gang det kommer en forespørsel.

6. Det er ingen overbooking.

Etterspørselsintensiteten til et rom til en bestemt pris på et fast tidspunkt i tid, oppgis som en etterspørselssannsynlighet. Ettersom etterspørselen etter de forskjellige romprisene varierer med hvor lenge det er igjen til siste salgstidspunkt, vil etterspørselssannsynligheten variere med tiden. Bookingperioden har jeg delt opp i flere beslutningsperioder, hvor det forekommer maksimalt en

etterspørsel i hver beslutningsperiode. Beslutningsperiode T er begynnelsen av bookingperioden, og beslutningsperiode 1 er siste salgstidspunkt. En forespørsel etter det dyreste hotellrommet vil alltid bli akseptert så lenge hotellet har ledige rom. Problemet er å bestemme om en skal akseptere eller avslå en forespørsel som ikke er til fullpris. Dette må en bestemme for prisklassene 2,3,4….,k, hvor k er antall prisklasser.

Den forventede inntekten ($V_t(r)$) til hotellet, hvis en forespørsel for prisklasse i blir akseptert i beslutningsperiode t, er lik prisen hotellet får for rommet som blir leiet ut, pluss den optimale forventede inntekten til hotellet i beslutningsperiodene t-1,……,0, når hotellet har r-1 rom igjen:

$$V_t(r) = I_i + V_{t-1}(r-1)$$

Hvis forespørselen blir avslått, blir den totale forventede inntekten lik den optimale forventede inntekten fra de gjenværende beslutningsperiodene t-1,….,0, men fortsatt med r ledige rom (ettersom en nå ikke har leiet ut noen rom):

$$V_t(r) = V_{t-1}(r).$$

En vil akseptere en forespørsel for pris klasse i, dersom den optimale forventede fortjenesten blir høyere ved å akseptere beslutningen enn å avslå den. Dette gir:

$$I_i + V_{t-1}(r-1) \geq V_{t-1}(r) \qquad \textbf{(4.2)}$$

eller

$$I_i \geq V_{t-1}(r) - V_{t-1}(r-1)$$

En har således et aksepteringskriterium for om en vil akseptere en forespørsel eller ikke. Dette aksepteringskriteriet kan vi finne matematisk ved å sette opp en rekursiv formel for den totalt forventede fortjenesten, $V_t(r)$. Denne formelen må inneholde både situasjonen hvor en avviser forespørselen, situasjonen hvor en aksepterer den, samt situasjonen hvor en ikke mottar noen

forespørsel. Sannsynligheten for at en ikke får noen forespørsel i beslutningsperiode t er gitt av P_{t0}. Den forventede fortjeneste til muligheten for å ikke motta noen forespørsel i periode t er lik $P_{t0}V_{t-1}(r)$. Hvis en mottar en forespørsel, vil beslutningen om en skal godta eller avslå forespørselen avhenge av hvilken prisklasse forespørselen er for. Som nevnt tidligere i oppgaven vil forespørselen alltid bli akseptert hvis den er for den dyreste prisklassen, så lenge det er ledige rom. Muligheten for en slik forespørsel for prisklasse 1 gir følgende forventede fortjeneste P_{t1} (I_1 + V_{t-1} (r – 1)) hvor P_{t1}er sannsynligheten for en forespørsel for prisklasse 1 i beslutningsperiode t. Hvis forespørselen er for en av prisklassene 2,3,4,….,k, må en bruke aksepteringsreglen (i (4.2)). Dette er gjort med følgende uttrykk:

$$\sum_{i=2}^{k} P_{ti} \text{ maks } (I_i + V_{t-1}(r - 1), V_{t-1}(r))$$

En får da følgende rekursiv formel:

$$V_t(r) = \begin{cases} P_{t0}V_{t-1}(r) + P_{t1}(I_1 + V_{t-1}(r-1)) \\ + \sum_{i=2}^{k} P_{ti} \text{ maks}(I_i + V_{t-1}(r-1), V_{t-1}(r)), & \text{for } r > 0, t > 0 \\ 0 & \text{ellers} \end{cases} \qquad \textbf{(4.3)}$$

Den forventede marginale verdien av et rom i beslutningsperiode t, gitt en gjenværende kapasitet på r ($\Delta V_t(r)$), er lik:

$$\Delta V_t(r) = V_t(r) - V_t(r-1).$$

Skriver vi om ligning (4.3) til å inneholde $\Delta V_t(r)$ får vi

$$V_t(r) - V_{t-1}(r) = \sum_{i=1}^{k} P_{ti} \text{maks}(I_i - \Delta V_{t-1}(r), 0) \qquad \textbf{(4.4)}$$

For utregningen av dette uttrykket, se Appendiks a
Uttrykket $V_t(r) - V_{t-1}(r)$ er kostnaden en har ved å holde alle rommene fra beslutningsperiode t til

beslutningsperiode t-1, når en har r rom. Fra (4.4) ser vi at det er mer økonomisk å leie ut det forespurte rommet enn ikke å leie det ut hvis første ledd i maksimeringsproblemet er positivt. En aksepterer forespørselen hvis fortjenesten en oppnår ved å akseptere er større eller lik den forventede marginale verdien av det enkelte rom. Vi kan utrykke denne aksepteringsreglen slik:

$$I_i \geq \Delta V_{t-1}(r) \quad \text{eller} \quad I_i \geq V_{t-1}(r) - V_{t-1}(r-1) \qquad \textbf{(4.5)}$$

Dette er den matematiske løsningen av den intuitive aksepteringsreglen foreslått i (4.2). Det er nå satt opp en DP modell. Nedenfor beskrives en algoritme av Lee og Hersh for å løse denne modellen (Lee & Hersh, 1993).

Lee og Hersh

Lee og Hersh viser at den forventede marginale verdifunksjonen $\Delta V_t(r)$ ikke øker med r for en fast t. Det vil si at for en fast beslutningsperiode vil vi ha lavere eller lik marginal romverdi pr rom jo flere rom en har igjen. De viser også at den forventede marginale verdifunksjonen $\Delta V_t(r)$ heller ikke avtar med t for en fast r. Eller sagt på en annen måte; for en gitt gjenværende kapasitet er verdien på et rom lavere eller lik jo nærmere en er siste mulige salgstidspunkt. Dette er logisk; jo kortere tid det er igjen til å få leiet ut rommene, desto mindre er muligheten for å få leiet ut det siste rommet til en dyrere prisklasse eller bare leiet det ut, og desto mindre er rommet verdt. I siste beslutningsperiode vil hotellet leie ut rommet til alle prisklasser.

For å illustrere dette, har jeg laget et eksempel med et hotell som har 3 prisklasser; henholdsvis 1500 kroner, 1000 kroner og 800 kroner. Sannsynligheten for å få en forespørsel i de tre prisklassene er som følgende:

Beslutningsperiode:	1-4	5-9	10-18	19-45
Ingen forespørsel	0,3	0,4	0,5	0,4
Prisklasse 1	0,2	0,2	0,1	0,1
Prisklasse 2	0,3	0,2	0,2	0,1
Prisklasse 3	0,2	0,2	0,2	0,4

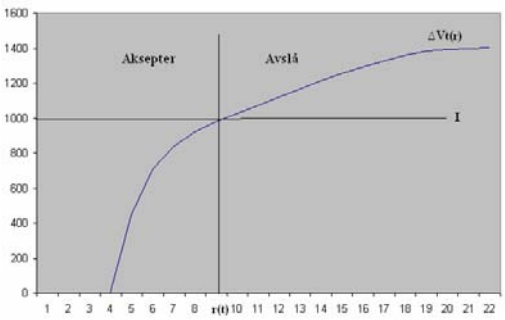
Ved hjelp av sannsynligheten til de forskjellige prisklassene, har jeg regnet ut alternativkostnaden ($\Delta V_t(r)$) til å leie ut rommet for ulik r når vi holder tidsperioden lik 12 og verdiene til alternativkostnaden $\Delta V_t(r)$ for ulike t når antall rom holdes lik 4. Dette er illustrert i figur 4.1. Ved å bruke monotonicyty til alternativkostnaden $\Delta V_t(r)$ vet vi at dersom vi aksepterer forespørselen på tidspunkt t for prisklasse i når det er k rom igjen, vil vi også akseptere forespørselen når det er flere enn k rom igjen. Det holder altså å finne det laveste antall rom som en er villig til å akseptere forespørselen med for hver prisklasse. I eksemplet nedenfor vil følgende terskler gjelde når vi holder tidsperioden fast på tidsperiode 12:

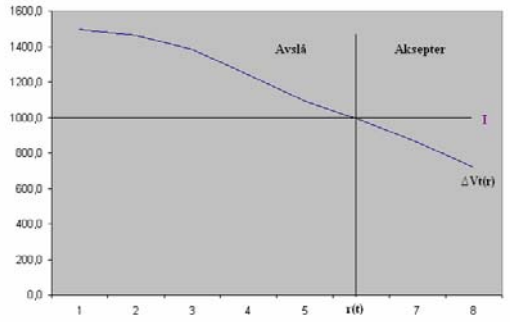
Prisklasse 1	aksepteres alltid så lenge det er ledige rom
Prisklasse 2	5 ledige rom
Prisklasse 3	8 ledige rom

Følgende terskler gjelder når vi holder antall rom fast på 4 rom:

Prisklasse 1	aksepteres alltid så lenge det er ledige rom
Prisklasse 2	tidsperiode 14
Prisklasse 3	tidsperiode 8

Dette kommer frem i Figur 4.1. Figuren viser at for noen verdier $r^*_i(t)$ når $r \geq r^*_i(t)$ er verdien av $\Delta V_{t-1}(r)$ større enn I_i og når $r < r^*_i(t)$ er verdien av $\Delta V_{t-1}(r)$ mindre enn I_i. I følge x.4 betyr dette at for $r < r^*_i(t)$ vil vi avvise forespørselen og for $r \geq r^*_i(t)$ vil vi akseptere forespørselen. En lik regel kan i tillegg lages for beslutningsperioder.





Figur 4.1 a) Kritisk beslutningsperiode **Figur 4.1 b)** kritisk booking kapasitet

Fig. 4.1.a) Denne figuren viser det kritiske booking nivået $t_2^*(4)$ for prisklasse 2 i eksemplet. Den viser at for $t > t_2^*(4)$ blir en forespørsel avslått og for $t \leq t_2^*(4)$ blir den akseptert.

Fig. 4.1.b) Denne figuren viser det kritiske booking nivået $r_2^*(12)$ for prisklasse 2 i eksemplet. Den viser at for $r \geq r_2^*(12)$ blir en forespørsel akseptert og for $r < r_2^*(12)$ blir den avslått.

For bevis av monotonicity av $\Delta V_t(r)$ og videre forklaring, se Lee og Hersh sin artikkel "A modell for dynamic airline seat invetory control with multiple seat bookings".

For å kunne ta en beslutning om å akseptere eller avslå forespørselen, er bare et sett av de kritiske verdiene nødvendige (enten antall rom eller tidsperiode). Dette settet av kritiske verdier vil bare bli endret når eller hvis etterspørselsforventningene blir endret. Endring av etterspørselsforventningene endrer etterspørselssannsynlighetene, som vil kunne endre hvilke forespørsler en aksepterer eller ikke (Lee og Hersh, 1993).

Bestilling av flere hotellnetter

Det er til nå kun sett på teori der det går an å bestille for en natt. Dette kan være relevant for flyplasshoteller, hvor det er vanlig å ikke bli boende mer enn en natt (ikke optimalt her ettersom det også er en del som bor flere netter). På de fleste hoteller i dag er det imidlertid vanlig at kundene bor mer enn en natt. Derfor vil det nå bli sett på en modell som tar hensyn til at noen kunder blir boende mer enn en natt. Denne teorien er hentet fra artikkelen "Revenue Management in a dynamic network enviroment" skrevet av Popescu og Bertsimas. Modellen deres er en statisk modell, men problemet som skal løses er et dynamisk problem. Jeg vil derfor kort gå gjennom det dynamiske problemet for flere netter, før jeg fortsetter med tilnærmingsmodellen til Popescu og Bertsimas.

Dynamisk programmeringsmodell for flere netter

Det dynamiske programmeringsproblemet for flere netter er det samme som for en natt. En vil akseptere en forespørsel for j netter for prisklasse i, dersom alternativkostnaden er lavere eller lik inntektene en vil få inn ved å leie ut rommet. Alternativkostnaden finner vi på følgende måte:

$$OC_i(n, t) = DP(n, t-1) - DP(n - A^i, T-1) \qquad \textbf{(4.6)}$$

Her er A^j de nettene som blir opptatt hvis en aksepterer forespørselen. Det er ikke mulig å bruke Lee og Hersh sin teori om threshold kapasitet dersom det er mulighet for å bestille rom for flere netter.

Popescu og Bertsimas

Å løse det dynamiske problemet kan være svært tidkrevende. Det vil derfor bli sett nærmere på artikkelen "Revenue management in a dynamic network environment" av Bertsimas og Popescu, hvor de foreslår en tilnærmingsalgoritme til den dynamiske modellen (4.6). I deres modell brukes en deterministisk tilnærming til det stokastisk dynamiske problemet. De går ut fra at den forventede etterspørselen til problemet er den etterspørselen som faktisk kommer. Når en vet hvilke forespørsler som kommer, kan vi finne den optimale løsningen på allokeringsproblemet ved hjelp av integer programmering. Ved å maksimere

$$IP(r, D_t) = \text{maks } I' \cdot y$$
$$\text{s.t} \qquad A \cdot y \leq r \qquad r \leq R$$
$$0 \leq y \leq D_t$$
$$y \text{ integer}$$

vil vi finne den optimale romallokeringen. Hvor $R = (R_1,\ldots,R_h)$, R_i er kapasiteten til hotellet for dag i og h er antall dager det kan bestilles rom på; $r = (r_1,\ldots,r_h)$ er den ledige kapasiteten på

tidspunkt t; $I = (I_1,\ldots,I_m)$ er priskategoriene, hvor I_j er prisen kunden må betale for å bruke s rom i h dager, for dagene k til v i prisklasse i; $y = (y_1,\ldots,y_m)$ hvor y_i er antall aksepterte forespørsler for s rom i h dager, for dagene k til v i prisklasse i og $A = (a_{11},\ldots.a_{1h}, a_{21},\ldots., a_{mh})$ hvor a_{ij} er antall hotellrom på dag j, som de som forespør priskategori i bruker (ettersom jeg i oppgaven har forutsatt at en kunde ikke etterspør mer enn et rom vil a_{ij} alltid være 1 eller 0). Vi vet også at $IP(D_t, r) \geq DP(t, r)$ ettersom vi med sikkerhet vet hvilken forespørsel som kommer (har vi determistisk forespørsel). Chen har vist at for hoteller vil $IP(r, D_t) = LP(r, D_t)$. Dette gjør at det er mye lettere å løse problemet og reoptimalisering går raskt. Vi kan altså løse integer-problemet ved hjelp av liner-programmering.

Hvis vi lar D_t stå for den forventede etterspørselen fra tidspunkt t og til siste salgstidspunkt for de ulike etterspørselsmulighetene, og dersom vi også går ut fra at D_t er den faktiske etterspørselen som kommer, har en for alle $r \leq R$ og $t \leq T$ et problem som kan løses med en LP modell. Modellen er

$$LP(r, D_t) = \text{maks } I' \cdot y \qquad \textbf{(4.7)}$$

$$\text{s.t} \qquad A \cdot y \leq r \qquad \textbf{(4.8)}$$

$$0 \leq y \leq D_t \qquad \textbf{(4.9)}$$

Hvor tiden t kan være beslutningsperioder, dataintervall eller andre tidsperioder spesifisert av brukeren. Målet i LP modellen er å maksimere inntektene og (4.7) gir inntektene en oppnår ved å allokere rommene. Konstrainten 4.8 utrykker at den gjenværende kapasiteten ikke må bli overskredet og 4.9 sikrer at antall allokerte rom ikke er under 0 og at antall utleide rom er mindre eller lik forventet etterspørsel for de gjenværende t tidsperiodene. $LP(r, D_t)$ er den maksimale verdien av r rom på tidspunkt t. For å vite om en skal akseptere eller avslå en forespørsel må en finne alternativkostnaden til det å akseptere forespørselen. Er denne alternativkostnaden lavere enn inntektene en får ved å akseptere forespørselen, velger en å akseptere forespørselen. For å finne alternativkostnaden til å leie ut et rom for en eller flere netter i tidsperiode t, bruker vi samme prinsipp som i DP modellen. Vi tar verdien i tidsperiode (t-1) og likt antall rom LP(r, t-1)

og trekker fra verdien av de ledige rommene som er igjen dersom en velger å leie ut til vedkommende som etterspør rom LP(r-Aj, t-1).

LP(r, t-1) - LP(r-Aj, t-1).

Vi får da følgende tilnærmingsalgoritme:

For alle kombinasjoner av gjenværende rom r og brukerspesifisert tid t

1. For en forespørsel etter klasse i på tidspunkt t med en gjenværende kapasitet på r rom, beregnes det LP-baserte estimatet av den forventede marginale verdien gitt av

   $\Delta V_1 (r) = LP(r, t-1) – LP(r-Ai, t-1)$

2. Aksepter forespørselen for klasse i hvis og bare hvis dens pris I_i er større eller lik forventet marginal verdi estimat:

   $I_i \geq \Delta V_1(r)$

3. Gå til steg 1 og fortsett så lenge som r>0 for hver t>1.

Merk at aksepteringsregelen i steg 2 er gitt som i Lee og Hersh metoden. Grunnen til at det ikke er nødvendig å regne verdifunksjonen i hver beslutningsperiode, er at verdien $V_tLP(r)$ er uavhengig av $V_{t-1}LP(r)$, i motsetning til verdifunksjonen i DP modellen. I algoritmen over tas det på hvert stadium en beslutning som er optimal, når den usikre etterspørselen som kommer for hver prisklasse er lik forventet etterspørsel D_t. Et problem med denne løsningsmetoden er at den bruker en deterministisk tilnærming. Den tar altså ikke hensyn til stokastisiteten i problemet. En utvidelse til dette algoritme er derfor å ta med variasjonen i etterspørselen. Dette kan gjøres ved å bruke Monte Carlo etterspørselsestimering (Bertsimas og Popescu, 2003).

Kapittel 5

I dette kapitlet går jeg gjennom hvordan dataprogrammet som brukes i kapittel 6 er laget. Jeg forklarer hvordan programmet er bygget opp og hvordan hver enkelt klasse fungerer. I gjennomgangen er det lagt vekt på å vise hvordan programmet løser oppgavene ut fra et økonomisk og matematisk perspektiv. Det er ikke lagt vekt på hvordan det er løst programmeringsmessig. Kildekoden til programmet er imidlertid vedlagt i appendiks g.

Dataprogrammet

For å kunne teste de ulike løsningsmetodene har jeg laget et dataprogram i programmeringsspråket Java. Programmet er laget for å kunne se på forskjell i inntekten til hotellet ved bruk av ulike løsningsmetoder. For å kunne regne ut hvilken inntekt hotellet får med de ulike metodene, må metodene få inn tilfeldige forespørsler etter rom. For å få de tilfeldige forespørslene stipulerer programmet disse selv. Det blir kun stipulert forespørsler for en natt, men forespørslene kan være for ulike prisklasser. Som forklart i kapittel 2, kan et dataprogram bestå av flere klasser. Dette dataprogrammet består av 9 klasser. En klasse er en del av programmet som løser en spesiell oppgave. For eksempel stipuleringsklassen som stipulerer en tilfeldig etterspørsel, eller revenuemanagementklassen som koordinerer alle de andre klassene. Programmet er laget slik at det blir stipulert 500 forskjellige etterspørselsrekker, og alle metodene regner ut inntekten for hver etterspørselsrekke. Hver metode regner dermed ut inntekten til hotellet 500 ganger, en gang for hver etterspørselsrekke.

For å kunne bruke dataprogrammet trenger vi å gi inn en del data til programmet. Disse dataene trenger programmet for å kunne stipulere en tilfeldig etterspørsel og for å kunne regne ut om forespørslene skal aksepteres eller ikke. En av de dataene som trengs er den forventede etterspørselen til alle prisklassene i etterspørselsperioden. Denne etterspørselsperioden gir dataprogrammet oss muligheten til å dele opp i flere dataperioder. Programmet åpner for denne muligheten for å kunne ta hensyn til at etterspørselen varierer over tid. En dataperiode er en del

av etterspørselsperioden det er samlet inn etterspørselsdata for, slik at vi kan finne den forventede etterspørselen til prisklassene i tidsperioden. Dataperiodene trenger ikke være av lik lengde. Et eksempel på variasjon i etterspørselen over tid er når vi har lav etterspørsel etter de dyreste prisklassene i starten av perioden og høy etterspørsel i slutten av perioden. Andre data som trengs i programmet er:

Antall rom på hotellet
Antall prisklasser
Prisen til de forskjellige prisklassene
Antall dataperioder
Hvor mange ganger programmet skal kjøres med disse dataene

Revenuemanagementklassen er hovedklassen i programmet. Den holder orden på de data som trengs og koordinerer de andre klassene. Når alle nødvendige data er på plass, starter den stipuleringsklassen. Stipuleringsklassen stipulerer en etterspørselsrekke. Den stipulerer når en forespørsel kommer og i hvilken prisklasse den kommer. Etter at forespørslene er stipulert, sendes etterspørselsrekken tilbake til revenuemanagementklassen. Revenuemanagementklassen sorterer forespørslene og andre nødvendige data, slik at de kan brukes av de andre klassene. Dataene sendes så til klassene optimaltsalg, EMSR, EMSRB og dynamisk programmering. Disse klassene avgjør så om forespørslene i etterspørselsrekken skal aksepteres eller avslås. Klassene bearbeider dataene og sender svar tilbake om forespørselen er akseptert eller ikke. Revenuemanagementklassen summerer opp inntektene til de ulike metodene ut fra hvilke av forespørslene som er blitt akseptert, og får totalinntekt til hotellet for hver metode. Nedenfor vil jeg gjennomgå prosessen i 5 av klassene. Jeg starter med stipuleringsklassen.

Stipulering av etterspørsel

Stipuleringsklassen stipulerer en tilfeldig etterspørsel etter å leie rom på hotellet. Etterspørselen stipuleres ved hjelp av den forventede etterspørselen til prisklassene. Denne etterspørselen representerer den virkelige etterspørselen etter rom i de ulike prisklassene. Etterspørselen antas å følge en poissen fordeling. Tilfeldighet er her viktig ettersom antall forespørsler,

etterspørselstidspunkt og prisklasse er ukjent.

I teorien om dynamisk programmering deles tidsperioden opp i flere beslutningsperioder. Dette gjøres også i stipuleringsklassen slik at de stipulerte forespørslene kan brukes på en dynamisk løsningsmetode. Ved valg av antall beslutningsperioder, må det derfor tas hensyn til at den dynamiske programmeringsmetoden kun er optimal når antall forespørsel pr. beslutningsperiode ikke overstiger en. Antall beslutningsperioder må derfor være så høyt at sannsynligheten for å få mer enn en forespørsel i beslutningsperioden er liten. Lar vi ϵ være et lite tall må følgende formel være oppfylt:

$P(r \geq 2) \leq \epsilon$ eller $1 - p(0) - p(1) \leq \epsilon$ (5.1)

Sannsynligheten for s forespørsler i beslutningsperioden finnes med følgende formel:

$P(s) = ((\mu_j/v_j)^s \exp(-\mu_j/v_j)) / s!$ for $s = 0, 1, 2,....$ **(5.2)**

hvor μ_j er det forventede antallet med forespørsler i dataperioden og v_j er antall beslutningsperioder i dataperioden. Formel 5.1 kan da ved hjelp av formel 5.2 skrives om slik:

$1 - \exp(-\mu_j/v_j) - (\mu_j/v_j)\exp(-\mu_j/v_j) \leq \epsilon$ **(5.3)**

For å finne antall beslutningsperioder v_j (som tilfredsstiller ligning 5.3) må vi ha antall forventede forespørsler μ_j i dataperioden. Antall forventede forespørsler finnes av klassen ved å summere de forventede forespørslene til alle prisklassene:

$\mu_j = \mu_{j1} + \mu_{j2} + ... + \mu_{jk}$

hvor μ_{ji} er forventet etterspørsel for prisklasse i, i dataperiode j. Stipuleringsklassen øker så antall beslutningsperioder v_j i ligning 5.3 med en, til ligningen er oppfylt. En kommer da frem til antall beslutningsperioder som brukes for dataperioden.

Sannsynligheten for at det kommer to eller flere forespørsler i en bestemt periode er mindre eller lik ϵ, men sannsynligheten for to eller flere treff i en av periodene er mye større. Har en 300 beslutningsperioder og en ϵ lik 0,001 er sannsynligheten for dobbel treff i en eller flere av periodene lik:

$$1 - (1-\epsilon)^{\text{antall beslutningsperioder}} = 1-(1-0{,}001)^{300} = 0{,}259$$

Når størrelsen på ϵ går ned, minker sannsynligheten for å få to eller flere forespørsler i en beslutningsperiode, men tiden det tar å regne ut svaret øker. Når en skal sette størrelsen på ϵ, må en derfor finne en balansegang mellom disse to.

Jeg har gått ut fra at etterspørselen til hotellet følger en poisson prosess. En poisson prosess kan simuleres ved å generere en sekvens av eksponentielt fordelte tilfeldige variabler, som representerer tidspunkt for når en kunde etterspør et hotellrom. De tilfeldige forespørselstidspunktene genererer stipuleringsklassen med følgende formel:

$$t_{ny} = t_{gamel} - \mu \cdot \ln (RND) \qquad \textbf{(5.4)}$$

hvor μ er gjennomsnittlig tid mellom forespørslene, t_{gamel} er tidspunktet til den siste forespørselen og RND er et uniformt distribuert tilfeldig nummer i intervallet [0,1]. For å kunne bruke formelen, må stipuleringsklassen finne gjennomsnittlig tid mellom forespørslene. Dette gjøres ved å ta lengden på dataperioden og dele på forventet antall forespørsler i dataperioden. Formel 5.4 blir så kjørt til t_{ny} er lik eller mindre enn siste tidspunkt i dataperioden. Dersom den stipulerte forespørselen kommer i en annen dataperiode (t_{ny} < siste tidspunkt i dataperioden) slettes denne forespørselen og programmet starter på ny med å stipulere forespørsler for neste dataperiode. Tidspunktene som fås på denne måten er i kontinuerlig tid, stipuleringsklassen gjør dem om til beslutningsperioder.

For å bestemme hvilken av prisklassene de enkelte forespørslene kommer i, finner klassen for alle prisklassene 1,2....,n, sannsynligheten for at forespørselen er i denne prisklassen. Sannsynlighetene finnes ved å ta forventet antall forespørsler for prisklasse i og dele på forventet

etterspørsel etter alle prisklassene 1,2..,n i dataperioden:

$P(i) = \mu_{ij}/\mu_j$

hvor μ_j er forventet etterspørsel til alle prisklassene i dataperiode j og μ_{ij} er forventet etterspørsel etter prisklasse i dataperiode j. Klassen gjør så sannsynlighetene(p_i) om til kumulative sannsynligheter (P_i) på følgende måte:

$P_1=p_1$, $P_2=p_1+p_2$, ….$P_N=p_1+p_2\ldots+p_n$.

P_i er sannsynligheten for en forespørsel i prisklasse i eller en dyrere prisklasse (jo høyere i jo billigere prisklasse). Siden sannsynligheten summerer seg til 1 er P_n lik 1. Når klassen har de kumulative sannsynlighetene, kan den stipulere hvilken prisklasse forespørselen er for. Dette gjør klassen ved å generere et uniformt distribuert tilfeldig tall mellom [0,1], og finner den første prisklassen som har en kumulativ sannsynlighet (P_i) lik eller større enn dette tallet.

EMSRA

Klassen EMSRA avgjør hvilke av forespørslene som skal aksepteres og hvilke som ikke skal aksepteres. For å avgjøre om en forespørsel skal aksepteres eller ikke brukes teorien til Beloba (EMSRa) som er gjennomgått i kapittel 4. Forespørslene kommer inn til klassen enkeltvis fra revenuemanagementklassen, slik de ville ha kommet i et virkelig scenario. EMSRAklassen regner så ut om forespørselen skal aksepteres eller ikke og sender svar til revenuemanagementklassen. Inntektene fra de aksepterte forespørslene blir summert i revenuemanagementklassen, og summen av disse er den totale inntekten til hotellet for den stipulerte forespørselsrekken ved bruk av EMSRAmetoden.

Jeg vil nå gå gjennom hvordan EMSRAklassen kommer frem til om forespørselen skal aksepteres eller avslås. For å kunne bestemme dette må klassen regne ut de kumulative

sannsynligheter $P_t(r)$, for alle r = (1,…, n) (Sannsynligheten for at den totale etterspørselen til prisklasse i er mindre enn eller lik antall rom reservert for prisklasse i, i tiden som er igjen til siste salgstidspunkt). Se Appendiks b for hvordan klassen regner ut de kumulative sannsynlighetene.

Når dette er gjort, regner EMSRAklassen ut reservasjonsnivået til alle prisklassene 1,..,k. Dette gjøres for prisklasse i (i = 1,2…n-1) ved å øke antall reserverte rom med 1 så lenge som:

Rompris prisklasse(i) · (1 - $P_t(r)$) ≤ rompris for prisklasse (i +1)

Når reservasjonsnivået til prisklasse i er funnet, kan bookinglimiten for prisklasse (i + 1) finnes ved å bruke formelen:

Bookinglimit(i + 1)=bookinglimit(i)-reservasjonsnivå(prisklasse i)

Bookinglimit for prisklasse 1 blir satt lik antall ledige rom på hotellet. Hvis bookinglimit til den forespurte prisklassen er større eller lik 1, blir forespørselen akseptert. Når en forespørsel aksepteres, reduseres antall ledige rom på hotellet med 1.

EMSRB

Klassen EMSRB er nesten lik EMSRAklassen. EMSRBklassen avgjør i likhet med EMSRAklassen hvilke av forespørslene som skal aksepteres og hvilke som ikke skal aksepteres. For å avgjøre om en forespørsel skal aksepteres eller ikke, brukes teorien til Beloba (EMSRB) som er gjennomgått i kapittel 4. Forespørslene kommer inn til klassen enkeltvis fra revenuemanagementklassen, slik de ville ha kommet i et virkelig scenario. EMSRBklassen regner så ut om forespørselen skal aksepteres eller ikke, og sender svar til revenuemanagementklassen. Inntektene fra de aksepterte forespørslene blir summert i revenuemanagementklassen, og summen av disse er den totale inntekten til hotellet for den stipulerte forespørselsrekken ved bruk av EMSRBmetoden.

Istedenfor å gå gjennom hele EMSRBklassen vil jeg her se på forskjellen mellom de to klassene.

Forskjellen på de to klassene er at EMSRA finner ett reservasjonsnivå for hver enkelt prisklasse som er dyrere enn den prisklassen vi skal finne bookinglimiten til, mens EMSRB regner ut et felles reservasjonsnivå for alle de dyrere prisklassene. Dette gjør at vi i EMSRBklassen ikke bruker ligningen:

rompris prisklasse(i) · (1 - $P_t(r)$) ≤ rompris for prisklasse (i +1) **(5.5)**

som blir brukt i EMSRAklassen. Men i stedet bruker ligningen:

$Pris_{1i}$ · (1 – $P_{1i}(r)$) ≤ rompris for prisklasse (i +1) **(5.6)**

Forskjellen på de to ligningene er at vi har byttet ut rompris prisklasse(i) og (1 - $P_t(r)$) i ligning 5.5, med $Pris_{1i}$ og (1 – $P_{1i}(r)$) i ligning 5.6. (1 – $P_{1i}(r)$) er sannsynligheten for at den totale etterspørselen til prisklasse i og alle dyrere prisklasser er større eller lik antall rom reservert for prisklasse i og alle dyrere prisklasser, i tiden som er igjen til siste salgstidspunkt. $Pris_{1i}$ er en fellespris for prisklasse i og alle dyrere prisklasser. $Pris_{1i}$ finner vi ved å ta prisen til hver prisklasse fra prisklasse 1 til i, multiplisere med prisklassens forventede etterspørsel og summere disse verdiene. Summen vi da får deles på summen av den forventede etterspørselen til prisklasse 1 til i. Dette gjøres slik:

$$I_{1,i} = \sum_{n=1}^{i} (I_n \cdot E(r_n)) / E(r_{1,i})$$

$$E(r_{1,i}) = \sum_{n=1}^{i} E(r_n)$$

Siden vi bruker et felles reservasjonsnivå for alle de dyrere prisklassene, blir også en forandring i hvordan vi finner bookinglimit. Istedenfor:

Bookinglimit(i + 1) = bookinglimit(i) – reservasjonsnivå (prisklasse i)

brukes:

Bookinglimit(i + 1) = C – reservasjonsnivå (prisklasse i)

hvor C er kapasiteten på hotellet. Grunnen til denne forandringen er at nå finnes et felles reservasjonsnivå for alle de dyrere prisklassene, og vi trenger da bare å trekke fra dette reservasjonsnivået fra kapasiteten for å finne bookinglimiten til prisklassen.

Dynamisk programmering

Klassen dynamisk programmering bruker teorien i artikkelen til Lee og Hersh, som er gjennomgått i kapittel 4. Klassen har som oppgave å akseptere eller avslå forespørslene som den mottar fra revenuemanagementklassen. Disse forespørslene kommer enkeltvis inn til klassen fra revenuemanagementklassen. Etter hvert som de enkelte forespørslene kommer inn avgjør klassen om hotellet skal akseptere eller avslå forespørselen. Er prisen som kunden er villig til å betale over eller lik alternativkostnaden aksepteres forespørselen, og er den under avslås den. Revenuemanagement klassen summerer så alle aksepterte forespørsler for å få den totale inntekten til hotellet.

Når klassen mottar den første forespørselen fra revenuemanagementklassen, regner den ut verdien $V_t(r)$ for alle mulige kombinasjoner av r (antall rom) og t (tidsperioden), hvor r = 1, 2, 3.....n og t = 1, 2, 3.....k(n er lik antall rom på hotellet og k er lik antall tidsperioder). Dette gjøres kun for den første forespørselen, ettersom disse verdiene lagres og kan brukes på de neste forespørslene. Nedenfor vil jeg gå gjennom hvordan klassen regner ut disse verdiene.

Når klassen skal regne ut verdien $V_t(r)$, tar den først for seg den siste dataperioden (dataperioden som er nærmest siste mulige salgstidspunkt). For denne dataperioden regner klassen først ut sannsynligheten for å få en forespørsel i tidsperiode t for prisklasse i. Utregning av

sannsynligheten for en forespørsel i prisklasse i, for tidsperiode t, blir funnet med følgende formel:

$P_t(i) = q_{it}/\text{antalltidsperioder}_t \cdot \exp(q_{it}/\text{antalltidsperioder}_t)$

hvor $\text{antalltidsperioder}_t$ er antall tidsperioder i dataperiode t og q_{it} er forventet etterspørsel etter prisklasse i, for dataperiode t. Disse sannsynlighetene regnes ut på ny for hver dataperiode, ettersom antall forespørsler i dataperiodene vil variere. Men sannsynligheten for å få en etterspørsel etter prisklasse i er lik for alle tidsperiodene som er innenfor samme dataperiode. Etter at sannsynlighetene er regnet ut, regner klassen ut verdiene i tidsperiode 1 ($V_1(r)$), for alle verdier av r og fortsetter med å regne ut verdien for tidsperiode 2 ($V_2(r)$, for alle verdier av r. Slik fortsetter programmet til det har regnet ut alle mulige kombinasjoner av $V_t(r)$.

For å kunne regne ut $V_t(r)$ må klassen først regne ut alternativkostnaden. Alternativkostnaden finner den ved formelen:

Alternativkostnaden = $V_{t-1}(r) - V_{t-1}(r-1)$.

I tidsperiode 1 settes alternativkostnaden automatisk til null siden den alltid er null i den første tidsperioden. Når klassen har regnet ut alternativkostnaden for tidsperiode t regner den ut

verdiene $V_t(r)$ for alle r i tidsperiode t. Dette gjøres på følgene måte:

$$V_t(r) = V_{t-1}(r) + \sum_{i=1}^{n} p_t(i) \cdot \text{maks}(\text{pris}\quad i - \text{alternativkostnaden}, 0)$$

(Rent teknisk gjøres dette på følgende måte i programmet: Programmet setter $V_t(r) = V_{t-1}(r)$. Når dette er gjort finner programmet høyeste verdi av (billigst pris - alternativkostnaden og 0). Programmet setter så $V_t(r)=V_t(r) + p_t(\text{billigst pris}) \cdot x_1$. Hvor x_1 er høyest verdi av (billigst pris - alternativkostnaden og 0) og $p_t(\text{billigst pris})$ er sannsynligheten for å få en forespørsel etter rom til billigst pris i tidsperiode t. Dette gjentas så for pris 2 hvor da programmet finner høyeste verdi

av (pris 2 - alternativkostnaden, 0) og $V_t(r)$ settes lik $V_t(r)=V_t(r) + p_t(pris2) \cdot x_2$. Dette gjentas til programmet har vært gjennom alle prisklassene.)

Når dette er gjort, starter den på ny med neste tidsperiode i dataperioden og fortsetter til den har tatt for seg alle tidsperiodene i dataperiodene. Disse utregningene gjøres kun en gang og det er når klassen får den første forespørselen.

Forespørslene kjøres så inn i programmet i kronologisk rekkefølge og aksepteres hvis prisen til prisklassen er høyere enn alternativkostnaden og avslås hvis den er lavere.

Lee og Hersh viste som tidligere nevnt i kapittel 4 at det ikke var nødvendig å regne ut alle verdiene til $V_t(r)$ på grunn av monotonitet. Jeg har av programmeringsmessige hensyn valgt å regne ut alle verdiene til $V_t(r)$. Programmet kunne derfor ha vært raskere men siden jeg ikke ser på tidsbruken til utregningene i denne oppgaven har ikke dette vært prioritert.

Optimaltsalg

Klassen optimaltsalg, regner ut den maksimale inntekten en kan få for hotellrommene, med forespørslene som kommer i tidsperioden. Dette gjøres ved å fylle opp rommene med de forespørslene som kommer for den dyreste prisklassen først. Hvis det er ledige rom igjen, fylles så disse opp med forespørslene som er for prisklasse 2 o.s.v, helt til alle rommene er fylt opp eller det ikke er forespørsel etter flere rom i tidsperioden. Klassen optimaltsalg kjenner til alle forespørslene før utregningen, og den vil derfor alltid fylle opp alle rommene så lenge det er større etterspørsel enn det er rom.

Kapittel 6

Testing av metodene

I dette kapitlet vil jeg presentere hvordan de ulike metodene er testet, samt resultatet av disse testene. Metodene dynamisk programmering, EMSRa og EMSRb er testet på et tenkt flyplasshotell. Metodene er først testet med et korrekt forecast, for så å bli testet med et feilaktig forecast. For testene med det korrekte forecastet er det laget 3 datasett, og metodene er kjørt 500 ganger på hvert sett. Årsaken til at det er laget mer enn ett datasett, er at det er viktig å ha flere datasett for å oppnå et generelt resultat. Dette er viktig fordi den metoden som gir best resultat for et datasett, ikke nødvendigvis gir beste resultatet for et annet datasett. I disse første testene ble altså metodene testet med rett forventet etterspørsel som input. I de påfølgende testene blir metodene testet for hvordan de gjør det med et feilaktig forecast. Med et feilaktig forecast menes et forecast som enten er for høyt eller for lavt i forhold til den reelle forventningen.

T-test

For å teste hvilken av metodene som gir høyest inntekt og kunne fastsette dette med stor grad av sikkerhet, har jeg valgt å bruke en t-test. Jeg har testet en og en metode opp mot hverandre i en paret t-test med et signifikansnivå på 99 prosent. De data som blir sammenlignet er naturlig paret. Hver av de 500 gangene programmet er kjørt, er inntektene kalkulert med de samme simulerte forespørslene. Forespørslene for hver metode er altså like, men bookinglimit og reservasjonsnivå blir satt av de ulike metodene. Forskjellen mellom inntektene som er oppnådd av de ulike metodene er funnet ved å trekke inntektene til den første metoden fra inntektene til den andre metoden. Dette er gjort for hver av de 500 gangene programmet kjører. La de to tilfeldige variablene (a_i, b_i) stå for inntektene oppnådd fra de to forskjellig metodene den i te gangen de kjøres, i = 1, 2,….,500, forskjellen i inntekt mellom de to metodene den i te gangen de kjøres er da lik:

$L_i = a_i - b_i$

Programmet regner ut L_i og lagrer L_i for hver av de 500 gangene programmet blir kjørt. Etter at programmet har stipulert et sett med forespørsler og regnet ut inntektene til de forskjellige modellene 500 ganger, regner det ut variansen og gjennomsnittet til den tilfeldige mengden. Mengden blir behandlet som en tilfeldig mengde på 500 fra en normal distribuert populasjon som har et gjennomsnitt på L. Er L=0 vil de to metodene gi lik inntekt i det lange løp, er L > 0 vil metode a gi høyere inntekt enn b i det lange løp. Tesen H_0 L=0 blir så testet mot tesen H_1 L>0.

Data i testen

De data som trengs for å kjøre metodene er:

Antall rom på hotellet

Antall prisklasser og prisen på hver prisklasse

Etterspørselsforventningen til hver prisklasse i dataperiodene

Verdien til ϵ

I testene av metodene er det brukt flere ulike romkapasiteter. På hotellet er det gått ut fra at størrelsen og standarden på rommene er lik, slik at det ikke har noe betydning for kunden hvilket rom han får.

Jeg har valgt å ha 4 prisklasser, hvor hver prisklasse dekker hvert sitt markedssegment. Markedssegmentene er klart adskilte fra hverandre, ved at de kundene som er i markedssegment i (i = 1,2,3,4) kun vil kjøpe prisklasse i. Jeg har gått ut fra at hotellet har klart å sette opp perfekte ”fences” mellom de ulike markedssegmentene slik at en verken har buy ups eller buy downs. Det vil si at en kunde som etterspør en prisklasse ikke vil velge en lavere prisklasse hvis den er

tilgjengelig, og heller ikke vil velge en høyere prisklasse hvis den prisklassen han forespør er utsolgt. Vi har altså en yieadeable etterspørsel. Prisene på hotellet er satt til 1400 kroner, 1200 kroner, 1000 kroner og 800 kroner.

Etterspørselsforventningene til de ulike prisklassene i dataperiodene kommer hotellet frem til ved bruk av forecast. Forecast er et estimat på den forventede etterspørselen frem til siste salgstidspunkt, basert på historiske observasjoner. Forecastet er også den informasjonen metodene bruker for å gjøre sine utregninger. I dette kapitlet har jeg brukt betegnelsene korrekt forecast og feilaktig forecast. Med et korrekt forecast mener jeg at forecastet er lik den forventede etterspørselen brukt i stipuleringsklassen for å stipulere forespørselene. Med et feilaktig forecast mener jeg at forecastet ikke er likt den forventede etterspørselen brukt av stipuleringsklassen. Jeg har i denne oppgaven ikke tatt stilling til hvilken metode hotellene bruker for å komme frem til forecastene. For en nærmere gjennomgang av problemstillingen med forecast viser jeg til kapittel 3.

Verdien til ϵ er i denne oppgaven satt til 0,01. En lavere ϵ vil som tidligere nevnt gi et bedre resultat for Lee og Hersh sin modell, men en lavere ϵ vil også føre til lengre utregningstid.

Etterspørselsdata som er brukt ved korrekt forecast

I dette avsnittet har jeg en gjennomgang av de etterspørselsdata som er brukt for å teste metodene når en har et korrekt forecast. Etterspørselsdataene består av den forventede etterspørselen til hver prisklasse for hver av de 6 dataperiodene. Den forventede totale etterspørselen til alle prisklassene er 102 rom for alle scenarioene. Datasettene blir kjørt mot 4 forskjellige romkapasiteter på hotellet. Henholdsvis 62, 82, 102 og 122 rom. Dette for å kunne se hvordan modellene virker med stor, liten og lik etterspørsel i forhold til romkapasiteten til hotellet. Totalt

består etterspørselsdataene av 3 datasett. Datasettene representerer 3 ulike scenarioer for hvordan etterspørselen etter de dyreste og de billigste prisklassene utvikler seg. I det første scenarioet reduseres den forventede etterspørselen etter de billigste prisklassene og øker for de dyreste prisklassene etterhvert som en nærmer seg den siste dataperioden. I det andre scenarioet øker den forventede etterspørselen etter de billigste prisklassene og reduseres for de dyreste prisklassene, etterhvert som en nærmer seg den siste dataperioden. I det tredje og siste alternativet har jeg gått ut fra at antall forventede forespørsler er jevnt fordelt i dataperiodene både for de billigste og de dyreste prisklassene. Datasettene kan ses i appendiks c.

Disse tre datasettene er valgt ut for å i størst mulig grad kunne dekke ulike etterspørselsmønstre som kan oppstå. Det første datasettet er nok den mest realistiske gjengivelsen av rometterspørselen ved et hotell, da den passer best overens med slik etterspørselen etter rom sannsynligvis vil være. Dette har jeg kommet frem til etter å ha snakket med Line Akerlind i Thon hotell og Tomm Caspersen på Grand hotell i Oslo. De jobber begge med revenue management for de nevnte hotellkjedene. De informerte om at forespørslene etter hotellrom kan komme så tidlig som ett år i forveien, men at hovedvekten av forespørslene kommer i slutten av salgsperioden. Etterspørselen etter de billigste prisklassene er størst i begynnelsen av salgsperioden og avtagende i slutten, mens de dyreste prisklassene hovedsakelig blir utleiet i slutten av salgsperioden. Antall forespørsler til de forskjellige prisklassene vil variere etter sesong, dag og hotell.

Testresultater

Testene ble utført ved å regne ut inntekten ved bruk av de tre metodene EMSRa, EMSRb og den dynamiske metoden. De tre datasettene og de 4 romkapasitetene vil gi 12 mulige kombinasjoner, og hver enkelt av disse kombinasjonene ble testet ut 500 ganger. For hver av de 500 gangene programmet ble kjørt, ble det stipulert en ny etterspørsel. Etterspørselen ble stipulert ved hjelp av den forventede etterspørselen til prisklassene i datasettet. Forespørslene ble gitt til metodene som regnet ut hvilken inntekt hotellet fikk ved bruk av den metoden. For hver gang inntekten ble regnet ut ble inntekten til metode i trukket fra inntekten til metode i+1 for å finne forskjellen mellom metodene. Alle de tre metodene ble sammenlignet med hverandre på denne måten. Dette

ble gjort for å finne variasjonen og forventningen på forskjellen i inntekt mellom metodene. Dette for å kunne utføre en paret T-test. Hvordan er utfører er paret T-test er forklart tidligere i kapitlet.

Resultatene av disse testene viser at vi med et signifikansnivå på 99 prosent kan si at den dynamiske metoden gir høyest inntekt, etterfulgt av EMSRb for alle datasettene som har en rom kapasitet på 102 rom eller færre. Med unntak av datasett 2 (hvor hovedvekten av forespørslene etter de dyreste prisklassene kommer i de siste dataperiodene) med en romkapasitet på 102 rom. Her kunne vi ikke med et signifikansnivå på 99 prosent, si at den dynamisk metoden er bedre enn EMSRb metoden.

Når de 3 datasettene hadde en romkapasitet på 122 rom, var det ikke mulig med et signifikansnivå på 99 prosent å si at en av metodene gav høyere inntekt enn noen av de andre metodene. Grunnen til dette er at en med 122 rom har høy kapasitet i forhold til antall forespørsler og det er derfor ikke nødvendig å avvise så mange av forespørslene.

Testene viser at den relative forskjellen i inntekten mellom de statiske metodene EMSRA og EMSRb og den dynamiske metoden øker når vi reduserer romkapasiteten. Dette skjer sannsynligvis på grunn av at de statiske metodene EMSRa og EMSRb ikke tar hensyn til den forventet etterspørselen til de prisklassene som vi ikke finner reservasjonsnivået til. Den forventet etterspørselen til disse prisklassene har også betydning for om det er optimalt å akseptere en forespørsel eller ikke. Er den forventede etterspørselen til den forespurte prisklassen stor, kan det lønne seg å ikke akseptere forespørselen, på grunn av stor mulighet for å leie ut rommet til samme pris på et senere tidspunkt. I mellomtiden har vi mulighet til å se hvordan etterspørselen etter de dyrere prisklassene utvikler seg. Om en bør vente eller ikke er blant annet avhengig av sannsynligheten for å kunne selge rommet igjen til samme pris på et senere tidspunkt. Når vi reduserer romkapasiteten, uten at dette påvirker forventet etterspørsel, øker sannsynligheten for å kunne leie ut rommet til samme pris eller en lavere pris på et senere tidspunkt. Dermed bør det reserveres flere rom til de dyreste prisklassene. Dette gjør den dynamiske modellen, men ikke de statiske modellene. Dermed øker den relative forskjellen mellom metodene. Når romkapasiteten blir lav nok, blir forskjellen mellom metodene mindre igjen. Dette skyldes sannsynligvis at forventet etterspørsel etter de dyreste prisklassene er så stor i forhold til romkapasiteten at nesten

alle rommene blir reservert til de dyreste prisklassene. Når alle metodene reserverer mesteparten av rommene til den dyreste prisklassen, blir forskjellen nødt til å bli mindre mellom metodene.

Grunnen til at de statiske metodene ikke tar hensyn til forventet etterspørsel til andre prisklasser enn de vi finner reservasjonsnivået til, er at metodene kun er laget for å regne ut reservasjonsnivået og bookinglimiten en gang. Uten mulighet for å endre på bookinglimit og reservasjonsnivå på et senere tidspunkt, har ikke den forventede etterspørselen til de andre prisklassene betydning for å finne det optimale reservasjonsnivået og bookinglimiten. Den forventede etterspørselen til disse prisklassene blir derfor ikke tatt hensyn til i de statiske metodene. Hvis modellen blir løst flere ganger etterhvert som tiden går, som vist i modell 4.1, vil reservasjonsnivå og bookinglimit kunne endres etterhvert som tiden går. Metodene blir dermed mer dynamiske, men de tar fortsatt ikke hensyn til muligheten for å kunne leie ut rommet til samme prisklasse på et senere tidspunkt. Økt mulighet til å leie ut rommene på et senere tidspunkt i samme prisklasse eller en lavere prisklasse, gjør at en bør reservere flere rom til de dyreste prisklassene.

Testene viser altså at med et korrekt forecast gir den dynamiske metoden høyest inntekt etterfulgt av EMSRb for alle romkapasitetene, med unntak av de romkapasitetene som er på 122 rom. Videre viser testene at de relative forskjellene mellom de statiske metodene og den dynamiske metoden øker når vi reduserer romkapasiteten. Sannsynligvis en følge av at de statiske metodene ikke tar hensyn til muligheten for å selge rommet til samme pris eller en lavere pris på et senere tidspunkt. Ut fra disse testene ser det derfor ut som om det vil lønne seg for hotellet å ha dynamisk prising og viktigheten av dette øker jo mindre kapasiteten er i forhold til etterspørselen. Testresultatene kan ses i appendiks d.

Testing av metodene ved bruk av feilaktig forecast

I sist avsnitt gikk jeg ut fra at vi hadde et korrekt forecast. I dette avsnittet går jeg ut fra at forecastet ikke er korrekt. Det vil si at metodene bruker et forecast som er feil. Et forecast kan for eksempel være på 33 rom selv om den reelle forventningen er på 30 rom. Metodene forventer da en poisson fordelt etterspørsel med et gjennomsnitt på 33 rom, men etterspørselen blir stipulert

med et gjennomsnitt på 30 rom. Forecastet er dermed 10 prosent for høyt. Metodene blir testet både med faste forecast feil og varierende forecast feil. En fast forecast feil vil si at forecast feilen er lik alle de 500 gangene programmet blir kjørt. For å få en fast forecast feil blir det korrekte forecastet ganget med (1+ forecast feilen i prosent), før sannsynlighetene blir regnet ut i de ulike metodene. Med varierende forecast feil vil forecastet være normal fordelt rundt den forventede etterspørselen (som er brukt til å stipulere forespørslene). Det normalfordelte forecastet blir testet med ulike standardavvik. For hvordan det normalfordelte forecastet blir stipulert se appendiks f. Metodene blir testet med normalfordelte forecast for å se om den dynamiske metoden også gir høyest inntekt ved et feilaktig normalfordelt forecast og lønnsomheten ved å forbedre forecastet. Metodene er testet med det faste feilaktige forecastet for å se hva som gir størst avvik mellom metodene. Ved de størst avvikene har jeg prøvt å forklare grunnen til avvikene.

Fast forecastfeil

For å teste metodene når en har en fast forecastfeil, har jeg brukt to etterspørselsscenarioer. I scenario 1 kommer etterspørselen etter de billigste prisklassene før de dyreste prisklassene, og det kommer kun etterspørsel etter èn prisklasse i en dataperiode (se tabell 6.1). Når forespørslene etter de ulike prisklassene kommer i en slik rekke følge, og det kun er mulig med en forespørsel pr beslutningsperiode, vil det ikke være forskjell på inntektene til modellene EMSRb og den dynamiske metoden. I testeksemplene er det 0,01 prosent sannsynlighet for at det kommer to eller flere forespørsel i en beslutningsperiode. Den dynamiske metoden tar ikke hensyn til muligheten for mer enn èn forespørsel i beslutningsperioden og vil derfor i enkelte tilfeller gi et annet resultat enn EMSRb. Hadde en redusert denne muligheten til 0,001 prosent ville forskjellen mellom metodene blitt mindre. Jeg har valgt å se bort fra denne forskjellen i forklaringen nedenfor, og bare konsentrert meg om forskjellen mellom EMSRA og den dynamiske metoden.

	Dataperiode 1	Dataperiode 2	Dataperiode 3	Dataperiode 4
Pris kl. 1	0	0	0	25
Pris kl. 2	0	0	25	0
Pris kl. 3	0	25	0	0
Pris kl. 4	25	0	0	0

Tabell 6.1 viser den forventede etterspørselen i de 4 dataperiodene. Dataperiode 4 er den dataperioden som er nærmest siste mulige salgstidspunkt.

I scenario 2 kommer alle forespørslene i samme dataperiode og den forventede etterspørselen er lik for alle prisklassene i dataperioden (se tabell 6.2). Scenarioene er testet med forecast som er 10, 20, 30, 50 og 60 prosent høyere enn et korrekt forecast for alle prisklassene, og et forecast som er 10 og 30 prosent lavere. Datasettene er testet på hotellkapasitetene 100, 80 og 60 rom for scenario 1 og 100 og 80 rom for scenario 2. Hvert datasett er kjørt 500 ganger, hver av de 500 gangene er inntektene til metodene regnet ut med det feilaktige forecastet og med korrekt. Resultatene til testingen med et fast forcast feil kan ses i appendiks e**.**

	Dataperiode 1
Pris kl. 1	25
Pris kl. 2	25
Pris kl. 3	25
Pris kl. 4	25

tabell 6.2 viser den forventede etterspørselen i dataperioden. All etterspørsel kommer i samme dataperiode.

Testresultater

Scenario 1

Testene viser som forventet at de feilaktige forecastene gav lavere inntekt enn det korrekte forcastet for alle de tre metodene, og jo mer feil forecastet er, jo lavere blir inntektene. Unntaket er EMSRA modellen som gav litt høyere inntekt for forecast som var 10 prosent høyere enn den forventede etterspørselen ved alle de tre romkapasitetene, se tabell 6.3. Dette skjer sannsynligvis fordi EMSRA ved utregning av bookinglimit ikke tar hensyn til alle prisklassene som er dyrere enn den prisklassen en finner booking limit til, og derfor reserverer for få rom ved bruk av korrekt forecast. På grunn av dette er ikke inntektstapet ved for høyt forecast i forhold til korrekt forecast så stort for EMSRA modellen som det er for de to andre modellene. For forecast som var 10 prosent for høye i forhold til den reelle forventningen gav EMSRA modellen høyere inntekt enn de to andre modellene, og for forecast som var enda høyere enn den reelle forventningen, gav EMSRA modellen enda høyere inntekt i forhold til de to andre modellene.

Ved forecastfeil som var lavere enn det korrekte forecastet, gav den dynamiske metoden og EMSRb modellen høyest inntekt i alle tilfellene.

Scenario 2

Testresultatene til scenario 2, når etterspørselen til de 4 prisklassene kommer i samme dataperiode, viser at både EMSRa og EMSRb gir høyere inntekt ved et litt for høyt forecast enn ved et helt korrekt forecast. EMSRa gav faktisk høyere inntekt for alle forecastene i testen som var høyere enn det korrekte forecastet ved en romkapasitet på 80 rom. Grunnen til at EMSR metodene gav høyere inntekt ved et litt for høyt forecast, er sannsynligvis at disse modellene ikke tar hensyn til muligheten for å få flere forespørseler i den prisklassen som en finner bookinglimit til og de prisklassene som er billigere. Når forecastfeilen var 30 prosent eller høyere, gav EMSR modellene høyere inntekter enn den dynamiske modellen for et hotell med en romkapasitet på 100 rom. For et hotell med en romkapasitet på 80 rom, gav EMSRb modellene høyere inntekt enn

den dynamiske modellen når forecastfeilen var 10 prosent eller høyere. EMSRA modellen gav høyere inntekt når forecastfeilen var 30 prosent eller høyere.

Ved forecastfeil som var lavere enn det korrekte forcastet gav den dynamiske metoden høyest inntekt i alle tilfellene, etterfulgt av EMSRb modellen.

Testing av metodene med varierende feilaktig forecast

Metodene testet med et normalfordeltforecast, er testet med standaravik på 10, 30 og 50 prosent av det korrekte forecastet. I dette tilfellet er metodene bare testet med et etterspørselsmønster. Metodene er testet ved at etterspørselen til alle prisklassene kommer i samme dataperiode og den forventede etterspørselen er lik for alle prisklassene i dataperioden(se tabell 6.2).

Testresultatene viser at den dynamiske metoden som gav høyest inntekt ved et korrekt forecast, ikke alltid gir den høyeste inntekten ved et varierende feilaktig forecast, se tabellene 2.2 (prosent) og 2.3 (reelle tall). Både EMSRa og EMSRb gir i enkelte tilfeller høyere inntekt enn den dynamiske metoden, avhengig av romkapasitet og standardavvik. Med for eksempel en romkapasitet på 100 rom og med standardavvik på 30 og 50 prosent, gir EMSRa metoden den høyeste inntekten av de tre metodene. Med en romkapasitet på 80 rom og standardavvik på 30 og 50 prosent gir EMSRb den høyeste inntekten, og ved en romkapasitet på 60 rom og standardavvik på 50 prosent gir også EMSRb den høyeste inntekten. Dette viser at den dynamiske metoden ikke nødvendigvis gir høyest inntekt når vi har et feilaktig forecast.

	100 rom	80 rom	60 rom
10	0,30 0,31, 0,02	1,48 1,97 0,50	1,51 2,17 0,67
30	-0,14 -0,98 -0,84	0,95 0,75 -0,20	1,27 1,64 0,38
50	-0,65 -2,01 -1,36	0,28 -0,67 -0,94	0,90 0,70 -0,21

Tabell 6.4 viser den prosentmessig forskjell i inntektene til hotellet med ulike forecastfeil og romkapasiteter. Det første tallet viser **EMSRa** i forhold til EMSRb, det andre tallet viser **EMSRa** i forhold til den dynamiske metoden og det siste tallet viser **EMSRb** i forhold til den dynamisk programeringsmetoden. Negativt fortegn betyr at den uthevede metoden har høyest inntekt.

	100 rom	80 rom	60 rom
10	106 327, 106 642, 106659	90 390, 91 744, 92 207	72 140, 73 244, 73 739
30	105 551, 105 404,104 529	**90 531, 91 403, 91 218**	71 818, 72 739, 73 014
50	105 438, 104 761, 103 356	**90 338, 90 587, 89 740**	**71 442, 72 093, 71 944**

Tabell 6.5 viser inntektene til hotellet med ulike forecastfeil og romkapasiteter. Inntektene til EMSRA kommer først, etterfulgt av EMSRb sine inntekter og til slutt dynamisk sine inntekter. Ved rød skrift har EMSRa gitt den høyeste inntekten av metodene, ved halvfet skrift har EMSRb gitt høyest inntekt og ved normal skrift har dynamisk gitt høyest inntekt

I tabell 6.6 ser en hvor mye bedre de ulike metodene gjør det med et perfekt forecast i forhold til et feilaktig, varierende forecast. Det mest oppsiktvekkende resultatet er at EMSRa og EMSRb i enkelte av tilfellene gjør det bedre med et feilaktig varierende forecast enn ved et helt korrekt forecast. F. eks ved forecast som har standardavvik på 30 prosent, gjør EMSRA modellen det 0,12 prosent bedre enn det korrekte forecastet.

	100 rom	80 rom	60 rom
10	-0,0075 -0,0034 0,0713	-0,0062 0,0589 0,3232	0,3499 0,1125 0,1991
30	0,0913 0,4789 1,4078	-0,1206 0,5444 1,5444	1,1057 1,0602 1,5208
50	0,4287 1,3340 2,7594	0,2431 1,5327 3,2685	1,6579 2,0102 3,1436

Tabell 6.6 viser i prosent forskjellen på det korrekte forcastet i forhold til det feilaktige forcastet. Første tallet viser EMSRa i forhold til korrekt forecast, andre tallet viser EMSRb i forhold til det korrekte forecasteog det siste tallet viser dynamisk programeringsmetoden i forhold til korrekt forecast. Negativt fortegn betyr at det feilaktige forecastet gjør det best.

Oppsummering av metodene

Testene viser at dynamisk programmering er den metoden som gir høyest inntekt ved bruk av et korrekt forecast, etterfulgt av EMRSb. Når kapasiteten er høy i forhold til etterspørsel, er valg av metode av mindre betydning siden hotellet da som regel vil ha ledig kapasitet. Testene viste også at jo lavere kapasiteten var i forhold til etterspørselen, desto større ble den relative forskjellen mellom metodene.

De fleste hoteller har nok ikke forecastprogrammer som beregner den forventede etterspørselen helt korrekt. I denne oppgaven har jeg derfor også testet metodene når hotellet ikke treffer helt perfekt med forecastet. I disse testene har jeg gått ut fra at hotellet sitt forecast er normalfordelt rundt den korrekte forventede etterspørselen. Testene brukt med et normalfordelt feilaktig forecast, viste at det ikke lenger var den dynamiske metoden som gav den høyeste inntekten i alle situasjonene. Ved høye standardavvik gav EMSR metodene høyere inntekt enn den dynamiske metoden for de dataene som var brukt i denne testen.

Resultatene fra disse testene viser at forskjellen i inntektene mellom metodene er avhengig av flere faktorer. Faktorer som spiller inn er etterspørselsmønster, romkapasitet i forhold til etterspørsel, og nøyaktigheten til forecastet. Andre faktorer som kan spille inn, men som ikke er testet i denne oppgaven, er antall prisklasser, prisen på prisklassene osv.

Appendiks a

$$V_t(x) - V_{t-1}(x) = P_0^t V_{t-1}(x) + P_1^t (F_1 + V_{t-1}(x-1))$$

$$+ \sum_{i=2}^{k} P_i^t \max(F_i + V_{t-1}(x-1), V_{t-1}(x)) - V_{t-1}(x)$$

$$= (1 - \sum_{i=1}^{k} P_i^t) V_{t-1}(x) + P_1^t (F_1 + V_{t-1}(x-1))$$

$$+ \sum_{i=2}^{k} P_i^t \max(F_i + V_{t-1}(x-1), V_{t-1}(x)) - V_{t-1}(x)$$

$$= V_{t-1}(x) - V_{t-1}(x) + P_1^t (F_1 + V_{t-1}(x-1))$$

$$- \sum_{i=1}^{k} P_i^t V_{t-1}(x) + \sum_{i=2}^{k} P_i^t \max(F_i + V_{t-1}(x-1), V_{t-1}(x))$$

$$= P_1^t (F_1 + V_{t-1}(x-1)) - P_1^t V_{t-1}(x)$$

$$+ \sum_{i=2}^{k} (P_i^t \max(F_i + V_{t-1}(x-1), V_{t-1}(x)) - P_i^t V_{t-1}(x))$$

$$= P_1^t (F_1 + V_{t-1}(x-1) - V_{t-1}(x))$$

$$+ \sum_{i=2}^{k} P_i^t \max(F_i + V_{t-1}(x-1) - V_{t-1}(x), V_{t-1}(x) - V_{t-1}(x))$$

$$= P_1^t (F_1 - \Delta V_{t-1}(x)) + \sum_{i=2}^{k} P_i^t \max(F_i - \Delta V_{t-1}(x), 0)$$

hvor det er brukt at $\sum_{i=1}^{k} P_i^t + P_0^t = 1$

Appendiks b

For å finne de kumulative sannsynlighetene $P_t(r)$ må EMSR klassen regne ut den forventede etterspørselen til prisklasse i, for alle prisklassene 1,….,k, for den tiden som er igjen til siste salgs tidspunkt. Den forventede etterspørselen til prisklasse i for tiden som er igjen til siste salgs tidspunkt finner klassen ved hjelp av følgende formel:

$$q_i = q_{ij} * (\text{tidsperioden} - \sum_{v=1}^{j-1} \text{antalltidsperioder}_v)/\text{antalltidsperioder}_j + \sum_{l=1}^{j-1} q_{il}$$

Hvor q_{ij} står for forventet forespørsel for prisklasse i, i dataperiode j, tidsperioden er den tidsperioden forespørselen kommer i, $\sum_{v=1}^{j-1} \text{antalltidsperioder}_v$ er antall tidsperioder i de dataperiodene som følger etter denne dataperioden, $\text{antalltidsperioder}_j$ er antall tidsperioder i dataperiode j og $\sum_{l=1}^{j-1} q_{il}$ er forventet etterspørsel i de dataperiodene som er igjen etter dataperiode j.

I neste steg regner EMSR klassen ut sannsynligheten ($p_t[r]$)for at nøyaktig **r** rom blir utleid. Dette blir gjort ved bruk av formelen:

$$p_t[r] = \exp(-q) * q^r / r!$$

Hvor r står for antall rom. Disse sannsynlighetene gjør EMSR klassen om til kumulative sannsynligheter $P_t(r)$. Sannsynligheten for at den totale etterspørselen til prisklasse i er mindre enn eller lik antall rom beskyttet for prisklasse i fra prisklasse j, i tiden som er igjen til siste

salgstidspunkt. Dette gjøres på følgende måte:

$P_t(r)= P_t(r-1)+ p_t[r]$

$P_t(1)$ settes lik $p_t(1)$. En får da at de kumulative sannsynlighetene (med stor P) er som følgende:

$P_1=p_1$, $P_2=p_1+p_2$, ….$P_N=p_1+p_2…+p_n$

Appendiks c

Datasett 1

	Periode 1	**Periode 2**	**Periode 3**	**Periode 4**	**Periode 5**	**Periode 6**
Pris kl. 1	0	0	0	1	4	7
Pris kl. 2	0	1	2	3	5	7
Pris kl. 3	15	8	5	4	3	1
Pris kl. 4	19	9	5	2	1	0

Datasett 2

	Periode 1	**Periode 2**	**Periode 3**	**Periode 4**	**Periode 5**	**Periode 6**
Pris kl. 1	7	4	1	0	0	0
Pris kl. 2	7	5	3	2	1	0
Pris kl. 3	1	3	4	5	8	15
Pris kl. 4	0	1	2	5	9	19

Datasett 3

	Periode 1	**Periode 2**	**Periode 3**	**Periode 4**	**Periode 5**	**Periode 6**
Pris kl. 1	2	2	2	2	2	2
Pris kl. 2	3	3	3	3	3	3
Pris kl. 3	6	6	6	6	6	6
Pris kl. 4	6	6	6	6	6	6

Appendiks d

Testresultatene med rett forecast vises i tabellene under. Resultatene fremkommer i prosentmessig forskjell. forskjellen i prosent mellom EMSRa og EMSRb er funnet ved at vi deler EMSRb på EMSRa, trekker fra 1 og ganger med 100. Forskjellen mellom EMSRa og den dynaniske metoden er funnet ved at vi deler den dynamiske metoden på EMSRa, trekker fra 1 og ganger med 100. Forskjellen mellom EMSRb og den dynaniske metoden er funnet ved at vi deler den dynamiske metoden på EMSRb, trekker fra 1 og ganger med 100. De resultatene hvor vi ikke kan bekrefte at den ene metoden er best med et signifikansnivå på 99 prosent er uthevet.

3 122 rom

	EMSRa	EMSRb	Dynamisk
EMSRa	0	**0,001163**	**0,0062019**
EMSRb	**0,001163**	0	**00,00504**
Dynamisk	**0,0062019**	**0,00504**	0
max	0,0227773	0,01899703	0,0139563

1 102 rom

	EMSRa	EMSRb	Dynamisk
EMSRa	0	0	0,10565134
EMSRb	0	0	0,10565134
Dynamisk	0,10565134	0,10565134	0
max	0,25348	0,25348	0,1476755

2 102 rom

	EMSRa	EMSRb	Dynamisk
EMSRa	0	0,3383167	0,389896799
EMSRb	0,3383167	0	**0,051406**
Dynamisk	0,389896799	**0,051406**	0
Max	1,5664916	1,22403	1,172025

3 102 rom

	EMSRA	EMSRB	Dynamisk
EMSRA	0	0,208569	0,303152789
EMSRB	0,208569	0	0,0943868
Dynamisk	0,303152789	0,0943868	0
Max	0,691996766	0,482421	0,3876687

1 82 rom

	EMSRA	EMSRB	Dynamisk
EMSRA	0	0,0116055595	0,5473181873
EMSRB	0,0116055595	0	0,53565046
Dynamisk	0,5473181873	0,53565046	0
max	1,015254347	1,00353232	0,465389

2 82 rom

	EMSRA	EMSRB	Dynamisk
EMSRA	0	2,0606295	2,476913407
EMSRB	2,0606295	0	0,407879
Dynamisk	2,476913407	0,407879	0
max	5,3957356	3,267769503	2,8485088675

3 82 rom

	EMSRA	EMSRB	Dynamisk
EMSRA	0	1,14455979	1,9736562
EMSRB	1,14455979	0	0,819714
Dynamisk	1,9736562	0,819714	0
max	3,021259	1,85546	1,027327

1 62 rom

	EMSRA	EMSRB	Dynamisk
EMSRA	0	0,1733701	0,814436
EMSRB	0,1733701	0	0,63995676
Dynamisk	0,814436	0,63995676	0
Max	0,9664952	0,7917524	0,1508304

2 62 rom

	EMSRA	EMSRB	Dynamisk
EMSRA	0	3,12771	3,604883
EMSRB	3,12771	0	0,4626986
Dynamisk	3,604883	0,4626986	0
max	6,309456596	3,085246	2,6104688

3 62 rom

	EMSRA	EMSRB	Dynamisk
EMSRA	0	2,2884944	3,3639617
EMSRB	2,2884944	0	1,0514059
Dynamisk	3,3639617	1,0514059	0
Max	4,2155222	1,8839145	0,8238466

Appendiks e

I dette appendikset er den prosentmessige forskjellen mellom et forcast med riktig forventning og et forcast med en feilaktig forventning oppgitt. Forskjellen er oppgitt i prosentmessig

Scenario 1 med en romkapasitet på 100 rom

Forcastet er 30 prosent for lavt

EMSR 0,1533
EMSRB 0,2616
Dynamisk 0,2616

Forcastet er 10 prosent for lavt

EMSR 0,083787
EMSRB 0,079168
Dynamisk 0,079168

Forcastet er 10 prosent for høyt

EMSR -0,00379
EMSRB 0,72579
Dynamisk 0,90771

Forcastet er 20 prosent for høyt

EMSR 0,4
EMSRB 4,0844
Dynamisk 4,0844

Forcastet er 30 prosent for høyt

EMSR	2,47853
EMSRB	9,42738
Dynamisk	9,42738

Forcastet er 50 prosent for høyt

EMSR	10,75789
EMSRB	14,51279
Dynamisk	14,40772

Scenario 1 med en romkapasitet på 80 rom

Forcastet er 30 prosent for lavt

EMSR	4,2393
EMSRB	5,8060
Dynamisk	5,7739

Forcastet er 10 prosent for lavt

EMSRA	2,3380
EMSRB	1,2700
Dynamisk	1,2700

Forcastet er 10 prosent for høyt

EMSR	-1,4132
EMSRB	1,1393
Dynamisk	1,4908

Forcastet er 30 prosent for høyt

EMSR	0,6609
EMSRB	5,1864
Dynamisk	5,1864

Forcastet er 50 prosent for høyt

EMSR	6,7399
EMSRB	14,9124
Dynamisk	13,8214

Scenario 1 med en romkapasitet på 60 rom

Forcastet er 30 prosent for lavt

EMSR	10,7889
EMSRB	7,5676
Dynamisk	7,2626

Forcastet er 10 prosent for lavt

EMSRA	3,5669
EMSRB	0,7116
Dynamisk	0,7088

Forcastet er 10 prosent for høyt

EMSR	-0,4304
EMSRB	0,7036
Dynamisk	0,7036

Forcastet er 30 prosent for høyt

EMSR	2,6481
EMSRB	9,0538
Dynamisk	9,0538

Forcastet er 50 prosent for høyt

EMSR	10,6338
EMSRB	11,2009
Dynamisk	11,2009

Scenario 2 med en romkapasitet på 100 rom

Forcastet er 30 prosent for lavt

EMSR	0,1923
EMSRB	0,2687
Dynamisk	0,2103

Forcastet er 10 prosent for lavt

EMSR	0,0464
EMSRB	0,0774
Dynamisk	0,0409

Forcastet er 10 prosent for høyt

EMSR	-0,0474
EMSRB	-0,10596
Dynamisk	0,011336

Forcastet er 20 prosent for høyt

EMSR	-0,06794
EMSRB	0,084771
Dynamisk	1,2245436

Forcastet er 30 prosent for høyt

EMSR	-0,17203
EMSRB	0,130164
Dynamisk	3,283509

Forcastet er 50 prosent for høyt

EMSR	-0,34152
EMSRB	3,173446
Dynamisk	6,42322

Forcastet er 60 prosent for høyt

EMSR	0,7831
EMSRB	4,7166
Dynamisk	7,2829

Scenario 2 med en romkapasitet på 80 rom

Forcastet er 30 prosent for lavt

EMSR	1,3979
EMSRB	1,9056
Dynamisk	1,7134

Forcastet er 10 prosent for lavt

EMSR	0,5201
EMSRB	0,6942
Dynamisk	0,3590

Forcastet er 10 prosent for høyt

EMSR	-0,59897
EMSRB	-0,58086
Dynamisk	0,308895

Forcastet er 30 prosent for høyt

EMSR	-1,811499
EMSRB	0,79788387
Dynamisk	0,95261

Forcastet er 60 prosent for høyt

EMSR	-1,6648
EMSRB	1,431918
Dynamisk	7,927916

Appendiks f

Det feilaktige forecastet blir stipulert ved først å finne sannsynligheten for de ulike forecastene x hvor x = (-5000, -4999,……,4999, 5000). Sannsynlighetene blir funnet ved formelen:

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

hvor μ er forventningen og σ er standardavviket. Sannsynlighetene blir så gjort om til kumulative sannsynligheter. For å stipulere det normalfordelte forcastet er det generert et uniformt distribuert tilfeldig tall mellom [0,1] og det første forcastet som har en kumulativ sannsynlighet (P_i) lik eller større enn dette tallet, blir gitt til metodene. Skule forecastet bli negativt blir det satt til 0.

Appendiks g

Kildekoden som er lagt med er brukt for å regne ut inntekten ved et korrekt forecast. Kildekoden som lager det feilaktig forecast er ikke tatt med for å redusere antall sider. En del av kilde koden for å lage det feilaktig forecast er dessuten veldig lik kilde koden til det korrekt forecast. Det kan hende det henger igjen litt kode etter det feilaktige forecastet.

```
package overbud;

import java.lang.*;
import java.io.*;
import java.util.*;


public class revenue {


    static BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

     stipuleringsinput[] stip_inputt;
     int antall_dataperioder;
     int antall_prisklasser;
     double[] prisklasser;

double total_inntekt_EMSR = 0;
```

```
double total_inntekt_EMSRb = 0;
double total_inntekt = 0;
double [] tidsperiode_forespørsel_test;
double type_forespørsel_test[] ;
int teller;
double tidsperiode_forespørsel[][]  ;
double[][] type_forespørsel ;
int []antall_forespørsler ;
double []antall_tidsperioder ;
double q_resterende_data_perioder[];
double q_resterende_data_perioderb[];
double q_resterende_data_perioder2[];
double q_resterende_data_perioderb2[];
double q[][];
double forcast__feil;
int starter=0;
int antall_forespørsler2;
int antallrom=40;

public static void main(String[] args)
      {
         Date date =new Date();
          int tid_1 = (int) date.getTime();
           new revenue();
          try
        {
 Date date2 =new Date();
  int tid_2 = (int) date2.getTime()-tid_1;
   reader.readLine();
       }
        catch (IOException io) {}
      }

      // konstruktor

 public revenue()
      {
      double totalinntekt_maksverdi=0;
      double totalinntekt_EMSR = 0;
      double totalinntekt_EMSRb = 0;
      double totalinntekt_utregning = 0;
      int antall_ganger_programet_kjøres = 500;
      double Forskjell_EMSRA_utregning[]=new double[antall_ganger_programet_kjøres];
      double Forskjell_EMSRA_EMSRB[]=new double[antall_ganger_programet_kjøres];
      double Forskjell_EMSRB_utregning[]=new double[antall_ganger_programet_kjøres];

      for (int i = 0; i<antall_ganger_programet_kjøres;i++)
```

```
{
   revenuemanagement2();
   totalinntekt_maksverdi=totalinntekt_maksverdi+ kjør_optimalt_salg();
   totalinntekt_EMSR=totalinntekt_EMSR+total_inntekt_EMSR;
   totalinntekt_EMSRb=totalinntekt_EMSRb+total_inntekt_EMSRb;
   totalinntekt_utregning=totalinntekt_utregning+total_inntekt;

   //statistikk t test
  Forskjell_EMSRA_utregning[i] = total_inntekt -total_inntekt_EMSR;
  Forskjell_EMSRA_EMSRB[i] = total_inntekt_EMSRb -total_inntekt_EMSR;
  Forskjell_EMSRB_utregning[i] = total_inntekt-total_inntekt_EMSRb;
  if (i==(antall_ganger_programet_kjøres-1))
  {
     double Gjenomsnittlig_forskjell_EMSRA_utregning=0;
     double Gjenomsnittlig_forskjell_EMSRA_EMSRB=0;
     double Gjenomsnittlig_forskjell_EMSRB_utregning=0;
     double Vaians_sample_EMSRA_utregning = 0;
     double Vaians_sample_EMSRA_EMSRB = 0;
     double Vaians_sample_EMSRB_utregning = 0;
     double summering_EMSRA_utregning = 0;
     double summering_EMSRB_EMSRA = 0;
     double summering_utregning_EMSRB = 0;

     for (int z=0;z<antall_ganger_programet_kjøres;z++)
      {
        summering_EMSRA_utregning=
        summering_EMSRA_utregning+Forskjell_EMSRA_utregning[z];
         summering_EMSRB_EMSRA=
        summering_EMSRB_EMSRA+Forskjell_EMSRA_EMSRB[z];
         summering_utregning_EMSRB=summering_utregning_EMSRB+Forskjell_EMS
        RB_utregning[z];
      }

Gjenomsnittlig_forskjell_EMSRA_utregning=summering_EMSRA_utregning/(antall_ganger_pr
ogramet_kjøres);

Gjenomsnittlig_forskjell_EMSRB_utregning=summering_utregning_EMSRB/(antall_ganger_pr
ogramet_kjøres);

Gjenomsnittlig_forskjell_EMSRA_EMSRB=summering_EMSRB_EMSRA/(antall_ganger_prog
ramet_kjøres);



      double sum_utregning_EMSRA = 0;
      double sum_EMSRB_EMSRA = 0;
      double sum_utregning_EMSRB = 0;
```

```
                for (int z=0;z<antall_ganger_programet_kjøres;z++)
                  {
                  sum_utregning_EMSRA =
                  sum_utregning_EMSRA+((Forskjell_EMSRA_utregning[z]-
                  Gjenomsnittlig_forskjell_EMSRA_utregning)*(Forskjell_EMSRA_utregning[z]-
                  Gjenomsnittlig_forskjell_EMSRA_utregning));
                  sum_EMSRB_EMSRA =
                  sum_EMSRB_EMSRA+((Forskjell_EMSRA_EMSRB[z]-
                  Gjenomsnittlig_forskjell_EMSRA_EMSRB)*(Forskjell_EMSRA_EMSRB[z]-
                  Gjenomsnittlig_forskjell_EMSRA_EMSRB));
                  sum_utregning_EMSRB=sum_utregning_EMSRB+((Forskjell_EMSRB_utregnin
                  g[z]-
                  Gjenomsnittlig_forskjell_EMSRB_utregning)*(Forskjell_EMSRB_utregning[z]-
                  Gjenomsnittlig_forskjell_EMSRB_utregning));
                  }
 Vaians_sample_EMSRA_utregning =
sum_utregning_EMSRA/(antall_ganger_programet_kjøres - 1);
 Vaians_sample_EMSRA_EMSRB = sum_EMSRB_EMSRA/(antall_ganger_programet_kjøres -
1);
Vaians_sample_EMSRB_utregning = sum_utregning_EMSRB/(antall_ganger_programet_kjøres
- 1);
System.out.println("Gjenomsnittlig_forskjell_EMSRA_utregning: " +
Gjenomsnittlig_forskjell_EMSRA_utregning);
System.out.println("Vaians_sample_EMSRA_utregning: " +
Vaians_sample_EMSRA_utregning);
 System.out.println("Gjenomsnittlig_forskjell_EMSRB_utregning: " +
Gjenomsnittlig_forskjell_EMSRB_utregning);
 System.out.println("Vaians_sample_EMSRB_utregning: " +
Vaians_sample_EMSRB_utregning);
 System.out.println("Gjenomsnittlig_forskjell_EMSRA_EMSRB: " +
Gjenomsnittlig_forskjell_EMSRA_EMSRB);
 System.out.println("Vaians_sample_EMSRA_EMSRB: " + Vaians_sample_EMSRA_EMSRB);
 }

}
 System.out.println("totalinntekt_maksverdi:   " +
(totalinntekt_maksverdi/antall_ganger_programet_kjøres));
 System.out.println("totalinntekt EMSR:   " +
(totalinntekt_EMSR/antall_ganger_programet_kjøres));
 System.out.println("totalinntekt EMSRb:   " +
(totalinntekt_EMSRb/antall_ganger_programet_kjøres));
 }

public void revenuemanagement2()
      {
       if (starter==0)les_inn_verdier_fra_konsoll();
```

```
starter++;
teller=0;
antall_forespørsler2=0;//flyttet
tidsperiode_forespørsel_test = new double[5000];
type_forespørsel_test = new double[5000];
tidsperiode_forespørsel  =new double[stip_inputt.length][5000];
type_forespørsel = new double[stip_inputt.length][5000];
antall_forespørsler = new int[stip_inputt.length];
antall_tidsperioder = new double[stip_inputt.length];
q_resterende_data_perioder=new double[antall_prisklasser];
q_resterende_data_perioderb=new double[antall_prisklasser];
q_resterende_data_perioder2=new double[antall_prisklasser];
q_resterende_data_perioderb2=new double[antall_prisklasser];
q = new double[antall_prisklasser][stip_inputt.length];


for (int index=0; index<stip_inputt.length; index++)
 {

   foresporselstipulering stip = new foresporselstipulering();
   stipuleringsresultat stip_result = stip.regn_ut(stip_inputt[index].antall_prisklasser,
   stip_inputt[index].q, index);

 for (int i=0; i<stip_result.antall_foresporsler; i++)
        {
       type_forespørsel_test[teller]=prisklasser[(int)stip_result.type_forespørsel[i]];
       teller++;
       tidsperiode_forespørsel[index][i]=stip_result.tidsperiode_forespørsel[i];
       type_forespørsel[index][i]=stip_result.type_forespørsel[i];
        }

       for(int a= 0; a<antall_prisklasser; a++)
            {
            q_resterende_data_perioder[a]+=stip_inputt[index].q[a];
            q_resterende_data_perioderb[a]+=stip_inputt[index].q[a];
            q_resterende_data_perioder2[a]+=stip_inputt[index].q[a];
            q_resterende_data_perioderb2[a]+=stip_inputt[index].q[a];
            q[a][index] = stip_inputt[index].q[a];
            }
       antall_forespørsler[index]=stip_result.antall_foresporsler;
       antall_tidsperioder[index]=stip_result.antall_tidsperioder;//
       antall_forespørsler2 +=  stip_result.antall_foresporsler;//

}

int Tidsperioder_gjennværende_dataperioder=0;
teller =0;
```

```
 for (int index=0; index<stip_inputt.length; index++)
 {
if(index==0)      for(int silje = stip_inputt.length; silje>index; silje--)
    {
      Tidsperioder_gjennværende_dataperioder +=antall_tidsperioder[silje-1];
    }
   Tidsperioder_gjennværende_dataperioder-=antall_tidsperioder[index];

    for (int i=0; i<antall_forespørsler[index]; i++)
           {
            tidsperiode_forespørsel_test[teller]=tidsperiode_forespørsel[index][i]+Tidsperiode
            r_gjennværende_dataperioder;
            tidsperiode_forespørsel[index][i]=tidsperiode_forespørsel_test[teller];
           teller++;

           }
 }
 Normalfordeling();
kjør_emsr();
kjør_emsrb();
kjør_utregning256();
}
double u2505 = 0;
double u2510= 0;
double u2515= 0;
double u2520= 0;
double u2525= 0;
double u2530= 0;
double u2535= 0;
double u2540= 0;
double u2545= 0;
double u2550= 0;
double feil;
Random random = new Random();
double [] kumulativ_sannsynlighet_for_etterspørsel_etter_x_rom= new double[50000];

public void kjør_emsr()
           {
           Date date =new Date();
           int tid_1 = (int) date.getTime();
           int antall_rom = antallrom;
           total_inntekt_EMSR = 0;
            for (int index=0; index<stip_inputt.length && antall_rom>0; index++)
            {

               for(int a= 0; a<antall_prisklasser && antall_rom>0; a++)
                        {
```

```
                q_resterende_data_perioder[a]-=stip_inputt[index].q[a];
                }

    for (int i=0; i<antall_forespørsler[index]&&antall_rom>0; i++)
    {

     EMSR emsr = new EMSR();
   double ny_gjest = emsr.hente_data_og_starte_emsr(q, prisklasser,
   antall_tidsperioder ,antall_rom, tidsperiode_forespørsel[index][i],
   type_forespørsel[index][i],q_resterende_data_perioder, index);
    if (ny_gjest > 0)
    {
       total_inntekt_EMSR += ny_gjest;
        antall_rom--;
      }
     }
  }
Date date2 =new Date();
int tid_2 = (int) date2.getTime()-tid_1;
 }


 public void kjør_emsrb()
        {
        Date date =new Date();
        int tid_1 = (int) date.getTime();
        int antall_rom = antallrom;
        total_inntekt_EMSRb = 0;
         for (int index=0; index<stip_inputt.length && antall_rom>0; index++)
         {
            for(int a= 0; a<antall_prisklasser && antall_rom>0; a++)
                        {
                        q_resterende_data_perioderb[a]-=stip_inputt[index].q[a];
                        }
           for (int i=0; i<antall_forespørsler[index] && antall_rom>0; i++)
           {
              EMSRb emsr = new EMSRb();
             double ny_gjest = emsr.hente_data_og_starte_emsr(q, prisklasser,
             antall_tidsperioder ,antall_rom, tidsperiode_forespørsel[index][i],
             type_forespørsel[index][i],q_resterende_data_perioderb, index);
            if (ny_gjest > 0)
           {
             total_inntekt_EMSRb += ny_gjest;
              antall_rom--;
            }
           }
         }
```

```
                Date date2 =new Date();
                int tid_2 = (int) date2.getTime()-tid_1;
               }



public void kjør_utregning256()
   {
      total_inntekt = 0;
      utregning256 stipu = new utregning256();
      int ledige_rom = stipu.hente_data_til_og_kjøre_utregning25(q, prisklasser,
antall_tidsperioder,   stip_inputt.length);

    for (int i = 0; i<antall_forespørsler2; i++)
        {
         if (stipu.akseptere_forespørsel_eller_ikke((int)type_forespørsel_test[i], ledige_rom,
         (int)tidsperiode_forespørsel_test[i]))
              {
              total_inntekt +=(int)type_forespørsel_test[i];
              ledige_rom--;
              }
        }
        }

public double kjør_optimalt_salg()
        {
        optimaltsalg topp =  new optimaltsalg();
        double max_verdi=topp.utregning(type_forespørsel_test,prisklasser,antallrom);
        return max_verdi;
        }

     private void les_inn_verdier_fra_konsoll()
     {
      try
      {
       System.out.println("Hvor mange prisklasser:");
       antall_prisklasser = Integer.parseInt(reader.readLine());
       prisklasser = new double[antall_prisklasser];
       System.out.println("Hvor mange dataperioder:");
       antall_dataperioder = Integer.parseInt(reader.readLine());
       stip_inputt = new stipuleringsinput[antall_dataperioder];
       for (int p=0; p<antall_prisklasser; p++)
       {
        System.out.println("Pris for prisklasse " + (p+1) + ":");
        prisklasser[p] = Integer.parseInt(reader.readLine());
       }
```

```
        for (int j=0; j<antall_dataperioder; j++)
        {
         double[] q = new double[antall_prisklasser];
         stipuleringsinput stip_input = new stipuleringsinput();
         for (int i=0; i<antall_prisklasser; i++)
         {
         System.out.println("Forventet foresporsel etter rom i dataperiode " +
        (antall_dataperioder-j) + " for prisklasse " + (i+1) + ":");
           q[i] = Integer.parseInt(reader.readLine());
           }
         stip_input.q = q;
         stip_input.antall_prisklasser = antall_prisklasser;
         stip_inputt[j] = stip_input;
        }
       }
       catch (IOException io) {}
       catch (NumberFormatException nf) {}
     }
    }

     class stipuleringsresultat
    {
     int antall_foresporsler;
     double antall_tidsperioder;
     double[] tidsperiode_forespørsel;
     double[] type_forespørsel;
    }

    class stipuleringsinput
    {
     int antall_prisklasser;
     double[] q;
    }

     class emsrresultat
    {
     double inntekt;
    }

    class foresporselstipulering
    {
     double nextRandomNumber[]=new double[9];
     int antall_dager_det_kan_bestilles_for=3;
     double [] sansynlighet_dette_antallet_med_dager = new
double[antall_dager_det_kan_bestilles_for];
     double [] sansynlighet_forespørsel;
     double [] antall_dager_forespørslen_er_for  = new double[5000];
```

```
double [] type_forespørsel = new double[5000];//brukes i Hvilke_forespørsel_er_det()
double[] Tidspunkt_forespørsel = new double[5000];
double[] Tidsperiode_forespørsel = new double[5000];
double Start_Tidspunkt = 24;
double antall_tidsperioder;//=antall_tidsperioder;//må hente antall tidsperioder
double lengde_Tidsenhet_i_tid;// = Start_Tidspunkt / tidsperiode;///antallTidsPerioder;
double Tid_ny = Start_Tidspunkt;
double snitt_tid_mellom_foresporsler; // = 2.4;//må regne ut selv // previously QQ
int index = 0; //fare endret fra 1
Random random = new Random();
int antallprisklasser;
double[] q;
double[] sannsynlighet_for_kunde_i_prisklasse;
double teller = Start_Tidspunkt;
double sumQ = 0;
int antall_foresporsler;

stipuleringsresultat regn_ut(int antall_prisklasser, double[] q,int index)
{
 stipuleringsresultat stip_resultat = new stipuleringsresultat();
 this.q = q;
 antallprisklasser = antall_prisklasser;
 faa_data_fra_konsoll();
   utregning_sannsynlighetlengdeTidsenhet_og_snitt_tid_mellom_foresporsler(); //
   kommenter denne bort og sett tidpseriode = 24 (f.eks.)
 sansynlighet();
 sansynlighet_antalldager();
 stipulering_foresporsel(index);

 stip_resultat.antall_foresporsler =  antall_foresporsler;
 stip_resultat.antall_tidsperioder = antall_tidsperioder;
 stip_resultat.tidsperiode_forespørsel = Tidsperiode_forespørsel;
 stip_resultat.type_forespørsel = type_forespørsel;
 return stip_resultat;
}

// constructor
foresporselstipulering() {}

public void stipulering_foresporsel(int dataperiode)
{

 int antall_foresporsel_tidspunkt = 0;
 while (Tid_ny > 0)
 {
```

```
  nextRandomNumber[0] = random.nextDouble();//flere ulike random for å få det til å bli
  random
  if (nextRandomNumber[0]>0.5)for(int u = 0; u<10;u++);
  nextRandomNumber[1] = random.nextDouble();
  if (nextRandomNumber[1]>0.5)for(int u = 0; u<10;u++);
  nextRandomNumber[2] = random.nextDouble();
  if (nextRandomNumber[2]>0.5)for(int u = 0; u<10;u++);
  nextRandomNumber[3] = random.nextDouble();
  if (nextRandomNumber[3]>0.5)for(int u = 0; u<10;u++);
  nextRandomNumber[4] = random.nextDouble();
  if (nextRandomNumber[4]>0.5)for(int u = 0; u<10;u++);
  nextRandomNumber[5] = random.nextDouble();
  if (nextRandomNumber[3]>0.5)for(int u = 0; u<10;u++);
  nextRandomNumber[6] = random.nextDouble();
  nextRandomNumber[7] = random.nextDouble();
  nextRandomNumber[8] = random.nextDouble();

  Tid_ny = Tid_ny + (snitt_tid_mellom_foresporsler *
  Math.log(nextRandomNumber[dataperiode]));
  Tidspunkt_forespørsel[index] = Tid_ny;
  antall_foresporsel_tidspunkt++;
  index++;
}
   antall_foresporsler = index-1 ; //har -1 her men vet ikke hvorfor, hindrer at siste
   forespørsel blir klasse 1//sansynligvis for at siste som blir laget av nytid er mindre enn 0
index = 0;//fare endret fra 1
int antall_foresporsler = 0;
int antallTidsperioder = (int)antall_tidsperioder;
while (antallTidsperioder  > 0)
{
   if((Tidspunkt_forespørsel[index]+999999999+999999999)>-999999999)  while
  (Tidspunkt_forespørsel[index] < teller )//if er for å kunne ha q lik null og det ikke blir
  uendlig løkke
  {
   teller = teller - lengde_Tidsenhet_i_tid;
   antallTidsperioder--;
  }
else antallTidsperioder=0;
  if (antallTidsperioder >= 0)
  {
   Tidsperiode_forespørsel[index] = antallTidsperioder;
   type_forespørsel[index]= Hvilke_forespørsel_er_det(dataperiode);
   antall_dager_forespørslen_er_for[index]= Hvor_mange_dager_er_forespørslen_for();
   antall_foresporsler++;
   index++;
  }
}
```

```
    }

  public int utregning_tidsperioder(double litenE, double q)  //skjekket og virker bra vis formel er
rett, får rett sumQ inn
    {
     int antallTidsPerioder = 0;
     double e = 1;
     while (e > litenE)
     {
      antallTidsPerioder++;
       e = 1 - Math.exp(-q/antallTidsPerioder) - q/antallTidsPerioder * Math.exp(-
       q/antallTidsPerioder);
     }
       return antallTidsPerioder;
    }

    public void utregning_sannsynlighetlengdeTidsenhet_og_snitt_tid_mellom_foresporsler()
    {
     for (int i=0; i<antallprisklasser; i++)
     {
     sumQ +=q[i];
     }
     antall_tidsperioder = utregning_tidsperioder(0.0001, sumQ);
     snitt_tid_mellom_foresporsler = Start_Tidspunkt / sumQ;
     lengde_Tidsenhet_i_tid = Start_Tidspunkt / antall_tidsperioder;
    }

    public void faa_data_fra_konsoll()
    {
     sansynlighet_dette_antallet_med_dager[0]= 0.3; //må skrives inn
     sansynlighet_dette_antallet_med_dager[1]= 0.2;  //må skrives inn
     sansynlighet_dette_antallet_med_dager[2]= 0.5; //må skrives inn
    }
   public void sansynlighet()//sansynligheten til den enkelte pris klasse**nytt navn** //sjekket
og  korrekt
        {
        sansynlighet_forespørsel = new double[antallprisklasser];
        for (int i = 0; i<antallprisklasser;i++)
           {
           sansynlighet_forespørsel[i]=q[i]/sumQ;
        if (i>0)
        sansynlighet_forespørsel[i]=sansynlighet_forespørsel[i]+sansynlighet_forespørsel[i-1];
           }
        }
   public int Hvilke_forespørsel_er_det(int dataperiode)  //Sjekket og stemmer
        {
```

```
        int w=0;
         nextRandomNumber[0] = random.nextDouble();//flere ulike random for å få det til å bli
         random
         if (nextRandomNumber[0]>0.5)for(int u = 0; u<10;u++);
        nextRandomNumber[1] = random.nextDouble();
        if (nextRandomNumber[1]>0.5)for(int u = 0; u<10;u++);
        nextRandomNumber[2] = random.nextDouble();
        if (nextRandomNumber[2]>0.5)for(int u = 0; u<10;u++);
        nextRandomNumber[3] = random.nextDouble();
        if (nextRandomNumber[3]>0.5)for(int u = 0; u<10;u++);
        nextRandomNumber[4] = random.nextDouble();
        if (nextRandomNumber[4]>0.5)for(int u = 0; u<10;u++);
        nextRandomNumber[5] = random.nextDouble();
        if (nextRandomNumber[3]>0.5)for(int u = 0; u<10;u++);
        nextRandomNumber[6] = random.nextDouble();
        nextRandomNumber[7] = random.nextDouble();
        nextRandomNumber[8] = random.nextDouble();

        for (int i = 0; i<antallprisklasser;i++)
            {
            if(sansynlighet_forespørsel[i]<= nextRandomNumber[dataperiode])
            {w = i+1;}
            }
           return w;
        }
    public int Hvor_mange_dager_er_forespørslen_for()//bestemmer hvor mange dager det blir
forespørt
        {
        int w=1;
            }
         return w;
        }
    public void sansynlighet_antalldager()//sansynligheten for hvor mange dager som blir bestilt
        {
        for (int i = 1; i<antall_dager_det_kan_bestilles_for;i++)
            {

sansynlighet_dette_antallet_med_dager[i]=sansynlighet_dette_antallet_med_dager[i]+sansynligh
et_dette_antallet_med_dager[i-1];
            }
        }
    }
```

```
class EMSR
 {
      double  tidsperioder;//=10;//må fås av stipulering   ** settes en gang
      double tidsperioden;//=10;//må fås av stipulering
      int antallprisklasser;
      double q;
      double[][] qq;// = new double [antallprisklasser]; //må fås av stipulering      ** settes en gang
      int  antallrom ; //må fås av stipulering   ** settes en gang
      double[] rompriser; //= new double [antallprisklasser];//må fås av stipulering  ** settes en
gang
      double[] sannsynligheter;
      double[] sannsynligheter_antallrom;
      int bookinglimit[]=new int[200];//tilfeldig lengde array her
      int pris_klasse;
      double[] rom_priser;
      double q_resterende_data_perioder;
      double q_resterende_data_perioder2[];
      int dataperioden;
      double tidsperioder_i_resterende_dataperioder;

 public void utregning_forventning()
           {
          q = q*(tidsperioden-tidsperioder_i_resterende_dataperioder)/tidsperioder
         +q_resterende_data_perioder;
           }


  EMSR() {}


  double hente_data_og_starte_emsr    (double[][] qqq, double[] rom_priser,  double[]
antall_tidsperioder, int antall_rom, double tidsperiode_foresporsel, double type_foresporsel,
double[] q_resterende_dataperioder, int dataperiode)//tall inn sjekket
                    {
                    this.rom_priser = rom_priser;
                    dataperioden = dataperiode;
                    q_resterende_data_perioder2=q_resterende_dataperioder;
                    tidsperioder = antall_tidsperioder[dataperioden];
                    tidsperioder_i_resterende_dataperioder = 0;
                    for(int i=dataperiode+1; i<antall_tidsperioder.length; i++)
                    {
                   tidsperioder_i_resterende_dataperioder =
                  tidsperioder_i_resterende_dataperioder+antall_tidsperioder[i];
                    }
```

```
                tidsperioden = tidsperiode_foresporsel;
                qq = qqq;
                rompriser = rom_priser;
                antallprisklasser = rom_priser.length;
                pris_klasse = (int)type_foresporsel;
                antallrom =antall_rom;
                sannsynligheter= new double[antallrom + 30];
                sannsynligheter_antallrom= new double[antallrom + 30];
                beregne_bookinglimit();
                return aksepter_forespørsel();
                }

public void beregne_bookinglimit()
                {
                for (int x=0;antallprisklasser>(x+1);x++)
                   {
                   q=qq[x][dataperioden];
                   q_resterende_data_perioder=q_resterende_data_perioder2[x];
                   utregning_forventning();
                   utregning_sannsynlighet();
                   int i = 0;

                   while((rompriser[x] * (1-sannsynligheter[i]))>rompriser[x+1] &&
                   i<=antallrom )
                            {
                            i++;
                            }

                   int reservasjonsnivå = i;
                   bookinglimit[0]=antallrom;
                   bookinglimit[x+1] = bookinglimit[x] - reservasjonsnivå;
                   }
                }

             public double regn_ut_fakultet(int i)//(*****) sjekket og fungerer
                      {
                         double t=i;

                      for(double c = i; c>1;)  //løkke for å regne ut !
                           {
                           c--;
                           t=t*c;
                           }
                      if(t==0) t=1;
                      return t;
                     }
```

```
    public void utregning_sannsynlighet() //(*****) sjekket og fungerer
                {

                for(int i= 0;i<=antallrom;) //løkke for å regne ut sannsynligheter
                   {
                   double fakultet=regn_ut_fakultet( i);
                   sannsynligheter[i] =Math.exp(-q) * (Math.pow(q,i)/fakultet);
                   i++;
                   }



                for(int w = 0; w<antallrom; w++)   //løkke for å finne kumulativ sannsynlighet
                   {
                   if (w<antallrom) sannsynligheter[w+1]+=sannsynligheter[w];
                   }
                }


          public double aksepter_forespørsel()
             {
             if (bookinglimit[pris_klasse]>0)
               {
               return rom_priser[pris_klasse];
               }
             else
               {
                return 0;
               }
             }
        }


  class EMSRb
  {
      //private static BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
       double  tidsperioder;//=10;//må fås av stipulering   ** settes en gang
       double tidsperioden;//=10;//må fås av stipulering
       int antallprisklasser;
       double q;
       double[][] qq;// = new double [antallprisklasser]; //må fås av stipulering      ** settes en gang
       int  antallrom ; //må fås av stipulering   ** settes en gang
```

```
      double[] rompriser; //= new double [antallprisklasser];//må fås av stipulering  ** settes en
gang
      double[] sannsynligheter;
      double[] sannsynligheter_antallrom;
      int bookinglimit[]=new int[200];//tilfeldig lengde array her
      int pris_klasse;
      double[] rom_priser;
      double q_resterende_data_perioder;
      double q_resterende_data_perioder2[];
      int dataperioden;
      double tidsperioder_i_resterende_dataperioder;
 public void utregning_forventning()
           {
           q = q*(tidsperioden-tidsperioder_i_resterende_dataperioder)/tidsperioder
+q_resterende_data_perioder;
           }

  EMSRb() {}


  double hente_data_og_starte_emsr    (double[][] qqq, double[] rom_priser,  double[]
antall_tidsperioder, int antall_rom, double tidsperiode_foresporsel, double type_foresporsel,
double[] q_resterende_dataperioder, int dataperiode)//tall inn sjekket
                    {
                    this.rom_priser = rom_priser;
                    dataperioden = dataperiode;
                    q_resterende_data_perioder2=q_resterende_dataperioder;
                    tidsperioder = antall_tidsperioder[dataperioden];
                    tidsperioder_i_resterende_dataperioder = 0;
                    for(int i=dataperiode+1; i<antall_tidsperioder.length; i++)
                    {
                   tidsperioder_i_resterende_dataperioder =
                   tidsperioder_i_resterende_dataperioder+antall_tidsperioder[i];
                    }
                    tidsperioden = tidsperiode_foresporsel;
                    qq = qqq;
                    rompriser = rom_priser;
                    antallprisklasser = rom_priser.length;
                    pris_klasse = (int)type_foresporsel;
                   antallrom =antall_rom;
                   sannsynligheter= new double[antallrom + 30];
                   sannsynligheter_antallrom= new double[antallrom + 30];
                    beregne_bookinglimit();
                    return aksepter_forespørsel();
                    }

public void beregne_bookinglimit()
```

```
{
for (int x=0;antallprisklasser>(x+1);x++)
   {
         double q2 = 0;
         for(int s = 0; (s-1) < x; s++)
         {
            q = qq[s][dataperioden];
            q_resterende_data_perioder = q_resterende_data_perioder2[s];
            utregning_forventning();
            q2=q+q2;
         }


   utregning_sannsynlighet(q2);

   int i = 0;

   while((gjenomsnitts_pris(x) * (1-sannsynligheter[i]))>rompriser[x+1] &&
   i<=antallrom )
            {
            i++;
            }
   int reservasjonsnivå = i;
   bookinglimit[0]=antallrom;
   bookinglimit[x+1] = antallrom - reservasjonsnivå;
   }
}


public double gjenomsnitts_pris(double finner_for_denne_prisklassen)
   {
   double antall_etterspørsler_reservasjonsnivå =0;
   double gjenomsnitts_pris =0;
   double total_pris_forventning = 0;
   double sum_verdi_prisklaser = 0;
                        for (int prisklasse = 0; (prisklasse-
       1)<finner_for_denne_prisklassen; prisklasse++)
    {
    sum_verdi_prisklaser=sum_verdi_prisklaser+rompriser[prisklasse];
    q=qq[prisklasse][dataperioden];
    q_resterende_data_perioder=q_resterende_data_perioder2[prisklasse];
    utregning_forventning();
        antall_etterspørsler_reservasjonsnivå =
       antall_etterspørsler_reservasjonsnivå + q;
    total_pris_forventning = total_pris_forventning + (rompriser[prisklasse] * q);
    }
```

```
            if(antall_etterspørsler_reservasjonsnivå>0)gjenomsnitts_pris =
            total_pris_forventning/antall_etterspørsler_reservasjonsnivå;
             else gjenomsnitts_pris=sum_verdi_prisklaser/(1+finner_for_denne_prisklassen);
             }


          public double regn_ut_fakultet(int i)//(*****) sjekket og fungerer
                    {
                   double t=i;
                   for(double c = i; c>1;)  //løkke for å regne ut !
                       {
                       c--;
                       t=t*c;
                       }
                       if(t==0) t=1;
                       return t;
                    }

    public void utregning_sannsynlighet(double qqqq) //(*****) sjekket og fungerer
                {
                 for(int i= 0;i<=antallrom;) //løkke for å regne ut sannsynligheter
                    {
                    double fakultet=regn_ut_fakultet( i);
                    sannsynligheter[i] =Math.exp(-qqqq) * (Math.pow(qqqq,i)/fakultet);
                    i++;
                    }
                    for(int w = 0; w<antallrom; w++)   //løkke for å finne kumulativ sannsynlighet
                    {
                    if (w<antallrom) sannsynligheter[w+1]+=sannsynligheter[w];
                    }
                }
       public double aksepter_forespørsel()
           {
            if (bookinglimit[pris_klasse]>0)
           {
        return rom_priser[pris_klasse];
            }
            else
           {
           return 0;
           }
           }
      }


class utregning256
{
```

```
    int antallprisklasser;
    int antallrom = 40;//totalt antallrom på hotellet
    double[][] verdien_for_dette_antalle_med_hotell_rom_i_denne_tidsperioden;
    double tidsperioder[];
    double[][] q;
    double[] prisklasser;
    double[] sannsynlighet_for_kunde_i_prisklasse;
    int tidsperiode =0;
    int dataperiode;
    int antall_dataperioder;
    double verdi_paa_alle_ledige_hotellrom = 0;
    double bidprice;


    private static BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));


public void utregning_sannsynlighet()
  {
   for (int i=0; i<antallprisklasser; i++)
      {
       sannsynlighet_for_kunde_i_prisklasse[i] = 1-Math.exp(-
q[i][dataperiode]/tidsperioder[dataperiode]);
       }
  }


  public int hente_data_til_og_kjøre_utregning25(double[][] qqq, double[] rom_priser,  double[]
antall_tidsperioder, int antall_dataperioder2)
      {
      antall_dataperioder=antall_dataperioder2;
      antallprisklasser = rom_priser.length;
      tidsperioder = antall_tidsperioder;
      q = new double[antallprisklasser][antall_dataperioder];
      q = qqq;
      sannsynlighet_for_kunde_i_prisklasse = new double[antallprisklasser];
      prisklasser =rom_priser;

      start();
      return antallrom;
      }

  public void start()
  {
```

```
   verdien_for_dette_antalle_med_hotell_rom_i_denne_tidsperioden = new double[antallrom
+2][(30*1000)];

   for(dataperiode = antall_dataperioder-1; dataperiode>=0; )
      {
     utregning_sannsynlighet();
       for (int tidsperiode_dataperiode = 1; tidsperiode_dataperiode < (tidsperioder[dataperiode] +
1); tidsperiode_dataperiode++)
         {
         tidsperiode++;

         }
    dataperiode--;
    }
 }

 public void setter_verdi_til_forskjellig_antall_ledige_hotellrom_i_perioden()
     {
            {
               if (antall_ledige_rom>0)bidprice =
               verdien_for_dette_antalle_med_hotell_rom_i_denne_tidsperioden[antall_ledige_ro
               m][tidsperiode-1] -
               verdien_for_dette_antalle_med_hotell_rom_i_denne_tidsperioden[antall_ledige_ro
               m-1][tidsperiode-1];
               verdi_paa_alle_ledige_hotellrom =
               verdien_for_dette_antalle_med_hotell_rom_i_denne_tidsperioden[antall_ledige_ro
               m][tidsperiode-1];
               regner_ut_verdi_på_ledige_hotell_rom();
                verdien_for_dette_antalle_med_hotell_rom_i_denne_tidsperioden[antall_ledige_r
                om][tidsperiode] = verdi_paa_alle_ledige_hotellrom;
               if (antall_ledige_rom == 0) {
               verdien_for_dette_antalle_med_hotell_rom_i_denne_tidsperioden[antall_ledige_ro
               m][tidsperiode] = 0; }
            }
     }

 public void regner_ut_verdi_på_ledige_hotell_rom()
         {
         for (int i=0; i<antallprisklasser; i++)
              {
              double x = Math.max(prisklasser[i] - bidprice, 0);
              verdi_paa_alle_ledige_hotellrom += (sannsynlighet_for_kunde_i_prisklasse[i] * x);
              }
         }

 utregning256(){}
```

```
public boolean akseptere_forespørsel_eller_ikke(int forespørselpris, int antallrom_ledige, int
tidsperiode)
  {
        if (antallrom_ledige>0 && tidsperiode>=0 &&
        forespørselpris>=verdien_for_dette_antalle_med_hotell_rom_i_denne_tidsperioden[antall
        rom_ledige ][tidsperiode]-
        verdien_for_dette_antalle_med_hotell_rom_i_denne_tidsperioden[antallrom_ledige-
        1][tidsperiode])
  {
   return true;
  }
  else
   {
   return false;
   }
  }
}
```

Referanseliste

Akerlind, Line revenue manager ved Thon Hotels

Belobaba, Peter P. *Application of a probabilistic decision model to airline seat inventory controll.* Operations Research, 37: 183-197, 1989.

Belobaba, Peter P. *Optimal vs. heuristic methods for nested seat allocation.* Proceedings of AGIFORS Reservations and Yield Management Study Group, side 28-53, 1992.

Bertsimas, Dimitris og Popescu, Ioana *Revenue management in a dynamic network environment.* Transportation Science, 37: side 257-277, 2003.

Bollapragada, Srinivas *Roundtable meeting at San Francisco, CA 12-13 november 2005* http://roundtable.informs.org/publc-access/min053a.htm

Boyd, Andrew E. og Kallesen, Royce. *The science of revenue management when passengers purchase the lowest available fare*. Journal of Revenue and Pricing Management 3, no. 2 (2004): 171-177.

Caspersen, Tomm revenue manager ved Grand Hotell

Chen, D. *Network flows in hotell yield management.* Technical report TR1225, Cornell University, 1998.

Cooper, William L. , Homem-de-Mello, Tito og Kleywegt, Anton J. *Models and the spiral-down effect in revenue management.* Informs,Vol. 54, No 5: side 968-987, 2006.

Zeni, Richard H.. *Improved Forecast Accuracy in airline revenue managemnet by unconstraining demand estimates from censored data*. AGIFORS, 2001.

Lee, Tak C og Hersh, Marwin *A model for dynamic airline seat inventory control with multiple seat bookings.* Transportation Science, 27: 252-265, 1993.

Littlewood, K. *Forecasting and control of passenger bookings.* AGIFORS, 12: 95-117, 1972.

McGill, Jeffrey I. og Van Ryzin, Garrett J. *Revenue Management: Research overview and prospects.* Transportation Science, Vol. 33, No 2, 1999.

Naughton, Patrick og Schildt, Herbert, *Java2 third edition.* Osborne/McGraw-Hill, 1999.

Weatherford, L. *A review of Optimization Modeling Assumptions in Revenue Management Situations.* AGIFORS, 1997.

Westermann, Dieter *(Realtime) dynamic pricing in an integrated revenue management and*

pricing environment: An approach to handling undifferetiated fare structures in low-fare markets. Journal of revenue and pricing management, volume 4, number 4, 2005.