

P2 - Simulering af Data

Gruppe B2-19

2020-04-21

Contents

Chapter 1

Introduktion

FIXME Husk link til Github.

Her vil vi gerne introducere rapporten

1.1 Forord

1.2 Abstract/Synopsis

1.3 Indledning

1.4 Anvendte pakker

Chapter 2

Problemanalyse

FIXME: Evt. emner: Fortæl om statistik. Fortæl om “almindelig” t-test og dens krav. Se, at resultater ikke giver mening, hvis krav ikke er opfyldt.

2.1 Statistik og historien bag

Ordet *statistik* stammer tilbage fra det latinske *statisticum collegium* (“statsrådsgiver”) og det italienske *statista* (“statsmand” eller “politiker”), (?). At statistik netop stammer derfra, giver god mening, under anskueelse af betydningen af disse to ord og den tidlige anvendelse af statistik. I takt med udviklingen af suverænitetssstaterne, steg behovet for at registrere befolkningen og dennes tilhørsforhold. Derfor anvendte statsrådsgivere og statsmænd statistik til at beskrive staten, særligt demografien. Senerere blev dette udvidet til at indsamle flere informationer, og ligeledes at analysere og fortolke disse ved hjælp af statistik.

Udgangspunktet for statistik kommer fra sandsynlighedsregning og læren derfra. Ligesom i sandsynlighedsregning arbejdes der i statistikken med udfald og tilfældigheder, men overordnet beskæftiger statistikken sig med analyse af indsamlet data.

To statistikerere som ligger fundamentet til den statiske arbejdsform der bruges i dag er Karl Pearson og Ronald Fisher. Karl Pearson var interesseret i at udvikle matematiske metoder til at studere biologisk arv og evolution. Gennem den interesse udsprang en række bidrag til statistiske anelyse metode. Pearson bidrog korrelationskoefficient (R^2) som bruges til at vise, hvor godt en regressionsmodel passer noget givent data. Udover det, lavede han også chi-i-anden-testen, som er en metode der bruges til at teste at ens observerede værdier stemmer overens med de forventede værdier, (?), (?).

Ronald Fisher designede et planteavl eksperiment der ultimativt gav mere

information, som krævede mindre tid, penge og anstrengelser. Han erfarede at hans data ikke var forventningsret, hvilket resulteret i upræcise og misledende resultater. Udfra dette introducerede Fisher randomisering. Princippet siger, at et eksperiment skal gentages på et antal kontrolgrupper, og de elementer brugt i eksperimentet skal tilfældig udvælges fra hele populationen. Dette gjorde data forventningsret, som mindsker effekten for variationen i et eksperiment. Udover det har Fisher bidraget med “Analysis of variance”, også kaldet ANOVA. Denne model er brugt til at analyserer forskellen mellem en samling af middelværdier i en stikprøve. (?)

Yderligere har udviklingen af computeren været med til at gøre anvendelsen af komplicerede statistiske beregninger hurtigere, mere præcise og mere tilgængelige. Anvendelsesområderne for statistik har ligeledes udviklet sig, fra i begyndelsen at være noget staten anvendte til styring af økonomi og befolkningsindblik, til stort set at være repræsenteret i alle større hverv i dag. Den moderne definition af statistik kan beskrives som evnen til at drage konklusioner om generelle tilfælde, *populationer*, på baggrund af enkelte tilfælde, *stikprøver*, (?, s. 1).

Statistisk videnskab giver tre overordnede begreber til anvendelsen. *Design*, som omhandler planlægningen og tilvejebringelsen af data til den undersøgelse der ønskes foretaget. *Beskrivelse*, når data er indsamlet skal der laves en sammenfatning over data og en indledende analyse, således der på et oplyst grundlag kan træffes beslutning om, hvorledes data skal behandles yderligere. *Inferens*, på baggrund af dataanalyse er det muligt at drage konklusioner om den population, der ønskes undersøgt. Forudsigelser, som laves på baggrund af data, kaldes for statistisk inferens, (?, s. 15-16).

2.2 Statistisk inferens

Det følgende afsnit er baseret på (?, ?) og (?).

Processen med at bruge dataanalyse til at drage konklusioner om en underliggende population ud fra en stikprøve af populationen, kaldes statistisk inferens. Dette er nyttigt, da det er hurtigere og billigere at indsamle og analysere data fra en delmængde af en population, end hvis der skulle indsamles de potentielt flere millioner observationer i populationen.

I statistisk inferens differentieres der mellem to metoder, *estimering* og *hypotese-test*. Når der estimeres på baggrund af en population, bruges stikprøven til at beskrive en ukendt del af populationen. Det kan f.eks være gennemsnitsindkomst af danskere, hvorved der findes et estimat $\hat{\mu}$ som bruges til at beskrive μ . Dette betegnes som et punktestimat, og vil oftest suppleres med et intervalestimat. Årsagen til dette er, at punktestimater i sin essens er tilfældig (det ændrer sig fra stikprøve til stikprøve). Derfor tilstræbes det at anvende et intervalestimat, hvor det kan siges at μ med 95% sikkerhed ligger, fremfor et punktestimat. Dette kaldes for et konfidensinterval.

Den anden form for statistisk inferens er hypotesetest. I hypotesetest opstilles først en nulhypotese, H_0 , og en alternativ hypotese, H_1 . Målet er at indsamle nok evidens imod H_0 , så den kan forkastes, og man er dermed blevet klogere på populationen. En nulhypotese kunne være, at danskere tjener det samme i gennemsnit som folk fra Sverige. Der vil ved hjælp af forskellige statistiske metoder, ses om der er en signifikant forskel i gennemsnittet. Hvis dette er tilfældet, vil nulhypotesen forkastes og den alternative hypotese vil antages at være sand.

2.3 Problemformulering

Kan simuleringsstudier bidrage til at højne kvaliteten af dataanalyser, og i så fald, hvordan?

1. Hvilke faldgruber skal man være opmærksom på ved hypotesetest?
2. Hvordan simulerer en computer data?
3. Hvordan kan teoretiske resultater eftervises ved hjælp af simulering?

FIXME: arbejdsspørgsmål i samme rækkefølge som i rapporten. Ret løbende.

Chapter 3

Simuleringer

3.1 Pseudorandom number generator

FIXME Mangler kilde

I dette afsnit introduceres begrebet *pseudorandom number generator* (PRNG) og hvorledes den kan bringes i anvendelse i forbindelse med simuleringer. I de tilhørende underafsnit beskrives først en type af PRNG kaldet lineær kongruens generator, og dernæst en metode til at omdanne uniformt fordelte talt til standard normalfordelte tal kaldet Box-Muller-transformation.

En computer fungerer ved, at den modtager et input, der bearbejdes af en algoritme, som derefter returnerer et output. Der findes ingen algoritmer, som er i stand til at generere faktisk tilfældige tal. Grunden til dette er, hvis der gives det samme input, til den samme algoritme, vil resultatet være det samme output som tidligere, fordi en computer fungerer på baggrund af matematik og derfor er deterministisk. Det er dog muligt ved hjælp af en beregningsmodel at skabe en illusion af ægte tilfældighed. Denne model kaldes *pseudorandom number generator* (PRNG).

En PRNGs funktionalitet beskrives ved nedenstående karakteristika.

1. Et input, kaldet seed, på baggrund af hvilket algoritmen beregner et pseudo-tilfældigt output. Herefter fortsætter algoritmen rekursivt, hvor det forrige output anvendes som nyt input.
2. Perioden, som beskriver hvor mange repetitioner algoritmen gennemløber, før outputtet begynder at gentage sig selv. Jo kortere perioden er, des mere gennemskuelig vil algoritmen fremstå.
3. Fordeling af de tal værdier der generes. Som det ses på figuren nedenunder, kan fordelingen af de genererede tal være jævn, hvilket viser en ligelig

fordeling af tallene. Ses der derimod et mønster eller tendens i fordelingen, vil algoritmen ikke være forventningsret, og alvendeligheden formindskes.

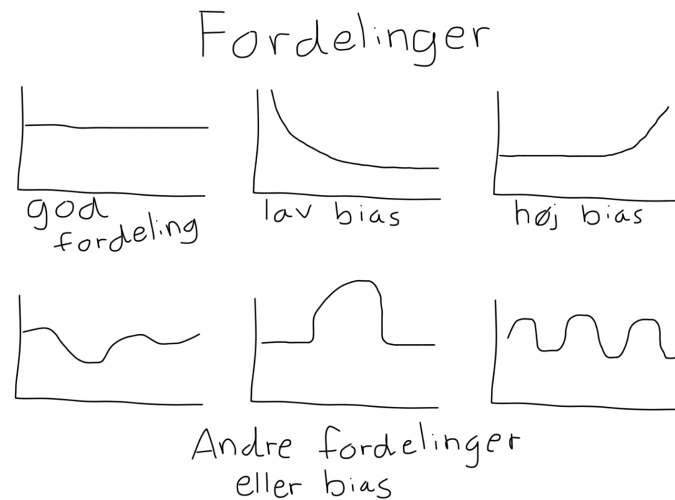


Figure 3.1: PRNG fordelinger

En af de tidligste algoritmer til PRNG, *middle-square method*, blev udviklet af John von Neumann. Svagheden ved denne metode er, at dens periode oftest er ret kort og derfor hurtigt begynder at gentage den samme talsekvens, (? , s. 12-13). I takt med udviklingen af de teknologiske hjælpemidler er der opstået mere effektive algoritmer til PRNG. En af de mere kendte og hyppigt anvendte algoritmer er lineær kongruens generator, som gennemgås i det følgende afsnit.

3.1.1 Lineær kongruens generator

Dette afsnit beskriver teoretisk hvorledes lineær kongruens generator på baggrund af en arbitrær parameter kan generere en sekvens af tilfældige tal, som kan være uniformt fordelte. Det følgende afsnit er primært baseret på kilden (?).

Lineær kongruens generator, på engelsk *linear congruential generator* (LCG), er en af de mange PRNG generatorer. Denne algoritme er en af de ældste og nemmeste at implementere. LCG danner en sekvens af tal ved iteration, og kræver kun få parametre. Helt specifikt er algoritmen angivet ved

Definition 3.1. Lineær kongruens generator danner en talsekvens,
 $X = [X_0, X_1, \dots, X_n]$, hvor $X_i = (a \cdot X_{i-1} + c) \bmod m$.
 Her er $a, c, m \in \mathbb{Z}$, og $0 < m$, $0 < a < m$ og $0 \leq c < m$.

Algoritmen fungerer således: Der startes ved en værdi angivet med X_0 , som bliver multipliceret med a og tilvæksten c lægges til. Derefter tages modulus,

m , af værdien. Modulus er en operator der dividerer et tal med et andet tal og returnerer restværdien.

I figuren nedenfor kan ses tre eksempler på LCG i aktion. De to første eksempler har samme parametre, men forskellige startpunkter.

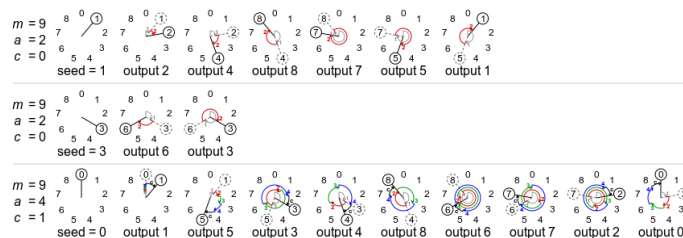


Figure 3.2: 3 eksempler på LCG. [lincongenvi]

I teorien er parameterne arbitrære, men i praksis bruges nogle værdier oftere end andre. Begrundelsen for dette er, at der findes “dårlige” værdier for parameterne, der vil returnere en tilsyneladende ikke-tilfældig sekvens af værdier. Et eksempel på dette er ved parameterne: $m = 64$, $a = 33$ og $c = 12$. Efter et antal iterationer vil der ses et mønster i værdierne, og det blotte øje vil altså kunne se at denne sekvens i virkeligheden ikke er helt tilfældig. Dette vil vises om lidt.

Eksempel

De følgende simuleringer er baseret på kilden (?), som også går mere i dybden med hvordan, der undersøges om ens parametre genererer tilsyneladende tilfældige værdier.

Først defineres en funktion for LCG algoritmen, som tager de nødvendige argumenter for algoritmen. Disse blev der redegjort for tidligere i afsnittet. Derudover angives også et argument, n , der fortæller hvor mange iterationer algoritmen skal foretage.

```
lcg <- function(m, a, c, seed, n) {
  #M er modulus, a er faktoren, c er tilvæksten, seed er
  #startværdien og n er antal iterationer.

  r <- numeric(n) #Definerer en ny variabel r,
                  #som er den numeriske værdi af n.
  r[1] <- seed #Angiver startpunktet.

  for (i in 1: (n-1)) {
    r[i + 1] <- (a * r[i] + c) %% m
  }
  return(r)
}
```

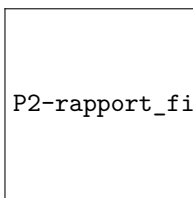
Det følgende er et eksempel på en LCG sekvens med en periode på 16. Perioden fortæller, hvor mange værdier generatoren returnerer inden den gentager sig selv.

```
lcg1 <- lcg(64, 33, 12, 57, 17)
lcg1
```

```
## [1] 57 37 17 61 41 21 1 45 25 5 49 29 9 53 33 13 57
```

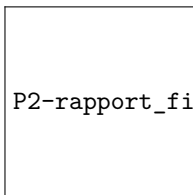
Plottes denne sekvens, vil det ses at værdierne ikke er tilfældige, da det er tydeligt se perioderne. Selv hvis antallet af gange LCG køres igennem ændres, vil det tydelige mønster ikke ændre sig.

```
lcg1_plot <- lcg(64, 33, 12, 57, 100)
plot(lcg1_plot, main = "Eksempel på en ikke tilfældig LCG sekvens",
     xlab = "Indeks", ylab = "Sekvens værdier", type = "l")
```




Nedenfor vises et eksempel på hvordan LCG algoritmen kan bruges til at lave en uniform fordeling i intervallet $[0, 1]$.

```
lcg2 <- lcg(86436, 1093, 0, 12, 1000)
hist((lcg2 + 0.05) / 86436, main = "LCG Uniform fordeling",
     xlab = "Sekvens værdier", ylab = "Frekvens")
```



Her kan det også nævnes, at hvis de værdier algoritmen returnerer plottes i et 2d-punkplot, vil der fremgå en tydeligere "tilfældig" fordeling, dog vil der stadig kunne ses et mønster.

```
plot(lcg2, main = "2d plot af LCG sekvens", xlab = "Indeks",
     ylab = "Sekvens værdier")
```




P2-rapport_files/figure-latex/unnamed-chunk-8-1.pdf

Afslutningsvist vil der vises et eksempel, hvor der returneres en sekvens der ser tilfældig ud.

```
lcg3 <- lcg(86436, 1093, 18257, 12, 100)

plot(lcg3, main = "'Tilfældig' LCG Sekvens", xlab = "Indeks",
      ylab = "Sekvens værdier", type = "l")
```



P2-rapport_files/figure-latex/unnamed-chunk-9-1.pdf

3.1.2 Box-Muller-transformation

I dette afsnit beskrives Box-Muller-transformation, hvilket er en metode til at generere standard normalfordelte tal ud fra uniformt fordelte tal. Dette gøres for at belyse, hvordan en computer kan generere tilsyneladende tilfældige tal, der er normalfordelt. Afsnittet er skrevet med inspiration fra (?). Metoden beskrives konkret i nedenstående sætning.

Sætning 3.1. *Box-Muller-transformation*

Antag, at U_1 og U_2 er uafhængige stokastiske variabler, der begge er uniformt fordelt på intervallet $[0, 1]$. Lad

$$Z_1 = \sqrt{-2\ln U_1} \cos(2\pi U_2) \quad \wedge \quad Z_2 = \sqrt{-2\ln U_1} \sin(2\pi U_2)$$

. Så er Z_1 og Z_2 uafhængige stokastiske variabler, der er standard normalfordelte.

Eksempel

I dette afsnit oprettes to normalfordelinger ved hjælp af Box-Muller-transformationen i R.

Først simuleres to populationer, U_1 og U_2 , hvor $U_1 \sim \text{unif}(0, 1)$ og $U_2 \sim \text{unif}(0, 1)$, som er de to uniformt fordelte populationer, der genereres normalfordelinger ud fra.

```
U1 <- runif(n = 100000, min = 0, max = 1)
U2 <- runif(n = 100000, min = 0, max = 1)
```

Disse to populationer benyttes nu til at oprette de to påstået standard normalfordelte populationer, Z1 og Z2.

```
Z1 <- sqrt(-2*log(U1))*cos(2*pi*U2)
Z2 <- sqrt(-2*log(U1))*sin(2*pi*U2)
```

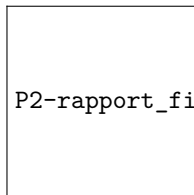
Efterprøvning i R

I det følgende afsnit efterprøves den forventede normalfordeling af Box-Muller-transformationen i R.

Det kan nu undersøges deskriptivt, hvilken fordeling Z1 og Z2 har.

Først oprettes et boksplot af de to populationer, der kan understøtte normalfordeling.

```
boxplot(Z1, Z2, main = "Boksplot af Z1 og Z2")
```



Det ses på boksplottet, at der er en indikation på normalfordeling med henvisning til de fire krav et boksplot af en normalfordeling opfylder. De fire krav er:

- 1) Median og middelværdi er lig hinanden.
- 2) Øvre og nedre kvartil er lige langt fra midten.
- 3) Der er lige mange outliers over øvre kvartil som under nedre kvartil.
- 4) 0,7 % af observationerne ligger som outliers.

Punkterne 2 og 3 tjekkes ved at betragte boksplottet, hvor der ikke umiddelbart synes at være noget, der modbeviser en normalfordeling af Z1 og Z2.

For at tjekke punkt 1, beregnes median (`median`) og middelværdi (`mean`) i kodelykket nedenfor.

```
mean_Z1 <- mean(Z1)
median_Z1 <- median(Z1)

mean_Z2 <- mean(Z2)
median_Z2 <- median(Z2)
```


Dette giver henholdsvis en middelværdi og median for Z1 på 2.4574776×10^{-4} og 3.7336997×10^{-4} , samt for Z2 på 0.003428 og 0.0042908. Disse resultater ligger meget tæt på hinanden, og den meget lave værdi stemmer overens med forventningen om, at Z1 og Z2 er standard normalfordelt.

For at tjekke punkt 4, beregnes andelen af outliers i hvert boksplot i kodestykket nedenfor.

```
OutVals1 <- boxplot(Z1, plot = FALSE)$out
outliers_Z1 <- length(OutVals1)

OutVals2 <- boxplot(Z2, plot = FALSE)$out
outliers_Z2 <- length(OutVals2)

outliers_andel_Z1 <- outliers_Z1/length(Z1)
outliers_andel_Z2 <- outliers_Z2/length(Z2)
```

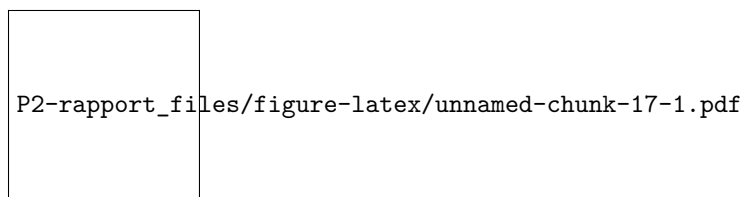
Dette giver en andel af outliers i Z1 på 0.00718 og i Z2 på 0.00726. Dette svarer til cirka 0,7 % outliers i både Z1 og Z2, hvilket stemmer overens med punkt 4.

Der er altså ikke noget evidens imod, at Z1 og Z2 skulle være normalfordelt. Tværtimod underbygger beregningen af deres middelværdi, at de er *standard* normalfordelte.

Hernest kigges der på histogrammerne for Z1 og Z2.

```
par(mfrow = c(1,2))
hist(Z1, main = "Histogram for Z1", ylab = "Frekvens", prob = TRUE)
lines(density(Z1), col = "blue")

hist(Z2, main = "Histogram for Z2", ylab = "Frekvens", prob = TRUE)
lines(density(Z2), col = "blue")
```



Det ses på histogrammerne, at de tilnærmelsesvist er normalfordelte, samt at de har en middelværdi på ≈ 0 og en standardafvigelse på ≈ 1 , og derved er Z1 og Z2 standard normalfordelte.

Desuden er det også en konsekvens af Box-Muller-transformationerne, at Z1 og Z2 er uafhængige. Dette undersøges ved hjælp af nedenstående graf.

FIXME: (Husk at indsætte figuren inden aflevering! :-))

Udover at eftervise uafhængigheden af Z1 og Z2 grafisk, kan det også undersøges ved hjælp af funktionen `cor` i R, som giver en værdi for korrelationen af de to talsekvenser. Korrelationen af to variabler udregnes ved formlen

$$\text{cor}(X, Y) = \frac{\text{cov}(X, Y)}{\text{sd}(X) \cdot \text{sd}(Y)}$$

Hvor *cov* angiver kovariansen og *sd* angiver standardafvigelsen.

Korrelation fortæller, hvor stor en lineær sammenhæng der er mellem to variabler. Denne værdi ligger i intervallet $[-1, 1]$, hvor -1 påviser en perfekt negativ lineær sammenhæng, 1 påviser en perfekt positiv lineær sammenhæng, og 0 viser, at der ingen lineær sammenhæng er.

```
cor(Z1, Z2)
```

```
## [1] -0.0004757555
```

Et resultat på afrundet -0.00048 , hvilket næsten er 0 , viser altså også at der ingen lineær korrelation er mellem Z1 og Z2, hvilket giver en indiktion for, at de er uafhængige.

3.2 Simulering af variabler

Simuleringer er generering af estimater på mulige udfald, og på den måde en efterligning af virkeligheden. Formålet er derved at generere tilfældigt estimerede værdier ud fra en model, der simulerer virkeligheden, hvilket muliggør yderligere analyser.

En definition på simulering er;

A situation or event that seems real but is not real, used especially in order to help people deal with such situations or events. - Cambridge Dictionary, (?).

Ud fra definitionen, er formålet altså ved simuleringer at efterligne virkeligheden, så de analyser der gøres på baggrund af simuleringerne, kan bruges i virkeligheden når lignende situationer opstår. Brancher hvor simuleringer er et yderst vigtigt redskab, er i motorsporten. I Formel 1 benytter holdene sig af simulatorer, hvor de genskaber bilerne og derved kan teste nye dele inden de producere dem i virkeligheden for at spare penge. Derved kan de analysere, hvilke forskellige dele der producerer mest *down-force*, uden at skulle teste dem i virkelig, (?).

3.2.1 Simulering i R

I dette afsnit gives eksempler på, hvorledes der kan udføres simuleringer af forskellige fordelinger, således disse fordelinger kan anvendes til statistisk inferens. Afsnittet er primært baseret på kilden (?).

Simuleringer i R udføres ved at estimere udfald fra en fordeling, hvor der bliver generet pseudo-tilfældige tal. Disse generede tal forekommer at være tilfældige, men er det reelt set ikke, (?).

Tallene er genereret ud fra et *seed*, som i R bestemmes ved `set.seed("værdi")`. Genereres tal ud fra samme *seed* vil værdierne være identiske.

Fordelinger, som der blandt andet kan simuleres i R, er normalfordelinger, binomialfordelinger og uniforme fordelinger.

En normalfordeling kan simuleres på følgende måde:

```
set.seed(1)
rnorm(10, mean = 0, sd = 1)

## [1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078 -0.8204684
## [7]  0.4874291  0.7383247  0.5757814 -0.3053884
```

Først sættes et *seed* så det er muligt, at rekonstruere samme simulering igen. Næste linje startes med, at skrive `rnorm` hvor “r” står for tilfældigt genererede tal, og “norm” for en normalfordeling. Inde i paranteserne angives antallet af værdier, der skal genereres, som bliver genereret ud fra en middelværdi på 0 og en standardafvigelse på 1. Eftersom middelværdien er 0 og standardafvigelsen er 1, kaldes denne normalfordeling for en standard normalfordeling eller Z-fordeling.

Ligeledes er det muligt at simulere binomialfordelinger og uniforme fordelinger:

```
set.seed(1)
rbinom(10, size = 2, prob = 0.5)
```

```
## [1] 1 1 1 2 0 2 2 1 1 0
```

Som før sættes et *seed*, og der skrives “r” før fordelingen der simuleres. Ligeledes er første værdi antallet af værdier der skal genereres. I binomialfordelingen er der angivet *size*, som er antallet repetitioner af succes/fejl ($1/0$), hvorimod *prob* er sandsynligheden for succes.

```
set.seed(1)
runif(10, min = 1, max = 2)
```

```
## [1] 1.265509 1.372124 1.572853 1.908208 1.201682 1.898390 1.944675 1.660798
## [9] 1.629114 1.061786
```

Den uniforme fordelingen er derimod angivet med en minimum- og maksimumværdi, hvor der genereres værdier imellem.

3.2.1.1 Sample og replicate

Derudover kan der simuleres på baggrund af observeret data. Ved at benytte funktionerne `sample` og `replicate` kan der dannes nye simuleringer. `sample` tager en stikprøve af den observerede data, hvor `replicate` kan gentage forskellige stikprøver.

```
set.seed(1)
Z_fordeling <- rnorm(1000, mean = 0, sd = 1)

Z_mean <- mean(Z_fordeling)
Z_mean
```

```
## [1] -0.01164814
```

Her eksekveres en Z-fordeling af 1,000 observationer hvor middelværdien, `Z_mean`, printes. Denne middelværdi vil variere afhængigt af det *seed*, der benyttes.

Dernæst kan der foretages en *sample* af variabelen `Z_fordeling`.

```
set.seed(1)
Z_sample <- sample(Z_fordeling, size = 10)

Z_sample
```

```
## [1] 1.1654620 0.2109073 -0.6816605 0.3439100 1.1594245 -0.8132443
## [7] -0.0505657 -0.9398293 0.5050676 1.4645873
```

```
mean(Z_sample)
```

```
## [1] 0.2364059
```

Her fremgår der 10 værdier, som er taget fra `Z_fordeling`, som gemmes i `Z_sample`. *Replace* gør så de værdier der tages og gemmes i `Z_sample` bliver lagt tilbage, og kan derved blive taget igen, så det er muligt at få den samme værdi flere gange.

Derefter er det muligt at gentage disse stikprøver ved brug af *replicate*

```
set.seed(1)
Z_replicate <- replicate(100, {
  x <- mean(sample(Z_fordeling, size = 10, replace = TRUE))
})
Z_replicate
```

```
## [1] 0.025838777 -0.203406868 -0.018837976 0.124819713 -0.164830739
## [6] 0.333733810 -0.423105712 -0.184935071 0.032046950 -0.166711138
## [11] 0.128328946 -0.003149890 0.120181675 0.114612739 0.196252732
## [16] -0.503558064 -0.057580606 0.448834352 0.302573714 0.349289426
## [21] 0.042763308 -0.449645007 -0.567764002 -0.246353993 0.241230288
## [26] 0.118362694 0.360972297 -0.502740430 -0.294227992 0.116454913
## [31] -0.458241385 0.145491729 0.468018395 -0.336889894 -0.033331555
## [36] 0.003202705 -0.230521260 0.002509074 -0.104908678 0.577232470
## [41] 0.405361935 0.571805412 -0.242967988 -0.070361799 -0.204749550
## [46] 0.529131107 0.123788480 0.428641140 0.421487366 0.296227471
## [51] -0.021464536 -0.563489156 -0.096866861 0.011460957 0.490675566
```

```
## [56]  0.118853950 -0.048831160 -0.433364372 -0.158288527  0.228563345
## [61]  0.383487805 -0.348257409  0.251253782  0.071396170  0.263839481
## [66] -0.659980071 -0.272080635 -0.065224329  0.133837717 -0.278633082
## [71] -0.543857905  0.066031905  0.675680619  0.172629193 -0.132092862
## [76]  0.269779598  0.150154779 -0.214823863 -0.098373108 -0.102126967
## [81] -0.350372760  0.502067558  0.111435486 -0.269840347  0.159325611
## [86]  0.033092712 -0.180166957  0.284144813 -0.108679711  0.028228077
## [91] -0.375649589 -0.224793309 -0.128662553  0.074568112  0.052000689
## [96] -0.061919916  0.245951870  0.073717038 -0.297133939  0.034445758
```

```
mean(Z_replicate)
```

```
## [1] 0.004120227
```

Her bliver en stikprøve af `Z_fordeling` foretaget 10x100 gange, hvorefter midelværdien på de 100 gentagelser bliver fundet.

3.3 Simulering af populationer

I det følgende afsnit simuleres der populationer, som i senere afsnit vil blive benyttet. Der simuleres to forskellige skæve fordelinger og en enkelt standard normalfordeling, hvorfra stikprøverne vil blive udtaget. Er det nødvendigt at benytte andre populationer, vil disse blive simuleret i de enkelte afsnit.

De to skæve fordelinger simuleres ud fra en betafordeling, $\text{Beta}(\alpha, \beta)$, hvor $\alpha - 1$ angiver antallet af succeser og $\beta - 1$ angiver antallet af fiaskoer. Betafordelingen er tilnærmelsesvis normalfordelt, hvis α og β begge er store eller omtrent ens, (?).

Der simuleres en højreskæv og venstreskæv population af størrelsen, $n = 10^4$. Den venstreskæve population har $\alpha = 8$ og $\beta = 2$, hvor den højreskæve population har $\alpha = 2$ og $\beta = 8$.

```
set.seed(1)
```

```
n <- 10000
```

```
pop_vs <- rbeta(n, shape1 = 8, shape2 = 2) # Venstreskæv population
pop_hs <- rbeta(n, shape1 = 2, shape2 = 8) # Højreskæv population
```

Figur ?? viser populationernes fordelinger.

Figure 3.3: To skæve populationer genereret fra en betafordeling

Chapter 4

Hypotesetest

I det følgende afsnit introduceres hypotesetests, samt hvorledes disse anvendes til at **FIXME** (beskrive/drage konklusioner for) en population, ved at opstille to hypoteser.

En hypotesetest baserer sig på det videnskabelige princip om falsificering. Der opstilles en indledende formodning om en population, kaldet nulhypotesen H_0 , og en alternativ, modsat hypotese H_1 . Er den indledende formodning ikke korrekt, må den alternative hypotese være gældende.

Ved en hypotesetest undersøges, hvorvidt der er en difference mellem observerede værdier og forventede værdier, hvis H_0 er sand.

Sandsynligheden for at der er en difference er stor, eftersom der arbejdes på en stikprøve og ikke selve populationen, og derfor benyttes et mål for, hvornår differencen er *for* stor, kaldet signifikansniveauet, α .

En hypotesetest viser sandsynligheden for mulige udfald, for på den måde at undersøge, hvorvidt H_0 kan forkastes, (?).

Et mål for, hvor usandsynlig en observeret værdi er, hvis H_0 er sand, kaldes for en teststørrelse.

For at kunne bestemme, om en difference mellem en observeret og forventet værdi er signifikant, benyttes en signifikanstest. En signifikanstest er en metode til at finde teststørrelsen og undersøge, om den er signifikant eller ej.

Teststørrelsen findes ofte som antallet af standardafvigelser, den observerede værdi, $\hat{\theta}$, ligger fra den forventede værdi θ_0 .

At $\hat{\theta}$ ligger mere end 3 standardafvigelser fra θ_0 , er højst usandsynligt, da $\hat{\theta}$ i så fald er en outlier i populationen. I et sådan tilfælde er θ_0 højst sandsynligt ikke populationens korrekte værdi.

En illustration af teststørrelsens betydning ved en normalfordeling kan ses på Figur ??.

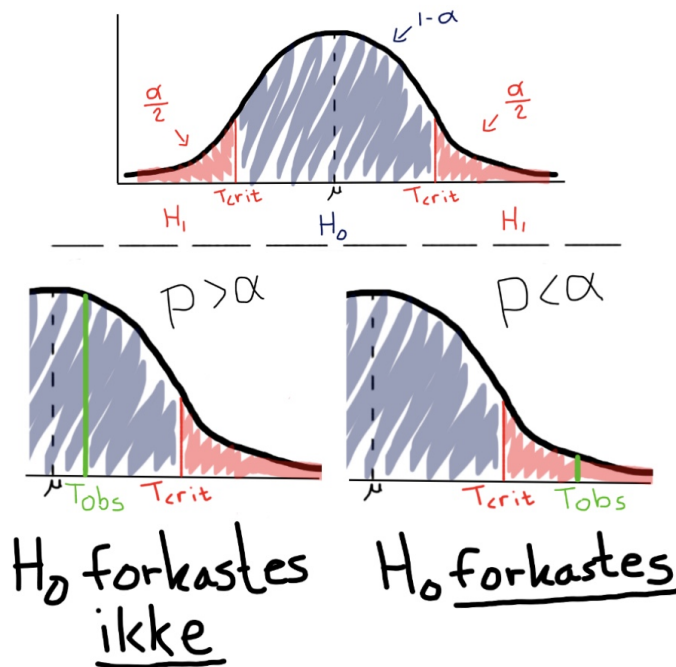


Figure 4.1: Teststørrelsens indflydelse på nulhypotesen

Derudover benyttes teststørrelsen til at udregne p -værdien, som er sandsynligheden for at få en teststørrelse, der er lige så eller mere ekstrem, hvis H_0 er sand.

Værdien af teststørrelsen påvirker p -værdien på den måde, at når teststørrelsen bliver mere ekstrem, falder p -værdien. Jo mindre p -værdien er, des mindre stoles på H_0 , og hvis p -værdien er mindre end signifikansniveauet, α , forkastes H_0 . Er p -værdien derimod større end α , er der ikke belæg for at forkaste H_0 - dette betyder dog ikke, at H_0 givetvis er sand.

Normalt arbejdes der med et signifikansniveau på 5%, $\alpha = 0.05$. Dog er der intet fast signifikansniveau og det kunne lige såvel være 10% eller 1%. Betydningen heraf diskuteres kort sidst i afsnittet under fejltypen, (?).

4.1 Hypotesetest for middelværdier (t-test)

I dette afsnit gennemgås fremgangsmåden for, hvordan en hypotesetest kan bruges til at bestemme middelværdien for en population. En sådan hypotesetest

kaldes en t-test.

Først er der nogle antagelser, der skal være opfyldt, for at t-testen ikke giver misvisende resultater.

1. Variablen er kvantitativ.
2. Stikprøveudtagning er udført med tilfældighed.
3. Populationen er normalfordelt.

Herefter opstilles hypoteserne. Nulhypotesen, $H_0 : \mu = \mu_0$ og den alternative hypotese $H_1 : \mu \neq \mu_0$.

Dernæst sættes et signifikansniveau, α , der vurderer med hvilken sikkerhed H_0 forkastes.

Derefter beregnes den observerede teststørrelse, $t_{obs} = \frac{|\bar{y} - \mu_0|}{se}$, hvor $se = \frac{s}{\sqrt{n}}$.

Til slut findes p -værdien, og på baggrund af denne, bliver nulhypotesen enten forkastet eller ej.

Eksempel

I dette afsnit gennemgås et eksempel på en t-test.

Følgende histogram viser en stikprøve af 10 observationer med en middelværdi på 0.1322028, udtaget fra en standardnormalfordelt population med en forventet middelværdi på 0.

Figure 4.2: Histogram over 10 simulerede standardnormalfordelte tal.

I kodelistykket nedenfor gennemgås den beskrevne fremgangsmåde for en t-test.

```
n <- 10

forventet_middelvaerdi <- 0 # Forventet middelværdi

middelvaerdi <- mean(xdata) # Middelværdi

standardafvigelse <- sd(xdata) # Standardafvigelsen

estimeret_standardfejl <- standardafvigelse/sqrt(n) # Estimeret standardfejl

t_obs <- (abs(middelvaerdi-forventet_middelvaerdi))/estimeret_standardfejl
# Observeret teststørrelse

alpha_halve <- 1 - pdist("t", q = t_obs, df = n-1, plot = FALSE)

p <- 2 * alpha_halve
```

Eftersom p -værdien er $0.6052327 > \alpha = 0.05$, forkastes H_0 ikke. Havde det derimod været en forventet værdi på -0.5 , ville p -værdien blive $0.0306291 < \alpha = 0.05$, hvilket vil medføre, at H_0 forkastes og det vil formodes, at H_0 ikke er korrekt for populationen.

4.2 Fejltyper

Der er risiko for to primære fejl når der foretages en hypotesetest. Den første, type-I fejl, er hvor H_0 forkastes, men i virkeligheden er sand, og den anden, type-II fejl, er hvor H_0 accepteres, men i realiteten er falsk.

En af de primære årsager til disse typer fejl er, hvor signifikansniveauet bliver sat, da dette i nogle tilfælde har stor betydning for, hvorvidt en hypotese bliver forkastet eller ej.

Sandsynligheden for type-I fejl er lig med det valgte signifikansniveau - i de fleste tilfælde 5%. Sandsynligheden for type-II fejl er derimod ikke let at præcisere. Dog er der stor sandsynlighed for type-II fejl, hvis den virkelige sandhed er tæt på nullhypotesen og lille, hvis den er langt fra. Ligeledes har stikprøvens størrelse indflydelse, eftersom meget data mindsker risikoen for type-II fejl, hvor der er større risiko ved mindre data, (?).

		Virkelighed	
		H_0	H_1
Konklusion	H_0	Korrekt	Type-II fejl
	H_1	Type-I fejl	Korrekt

Figure 4.3: Tabel over fejltyper

FIXME: Skriv om 'styrken' (jævnfør arbejdsblad fra 26/3)

4.3 Uparret t-test for ikke-normalfordelt stikprøve

Foretages en t-test på en ikke-normalfordelt stikprøve, kan resultaterne ikke altid antages at være retvisende. Dette vil nu vises. Først udtrækkes to stikprøver på de skæve populationer i afsnit ??.

```
#To stikprøver trukket fra populationerne
Stik1 <- sample(pop_hs, size = 100)
Stik2 <- sample(pop_vs, size = 100)
```

Figure 4.4: Stikprøve fra højreskæv betafordeling hvor $n = 100$, $\alpha = 2$, $\beta = 8$

Figure 4.5: Stikprøve fra venstreskæv betafordeling hvor $n = 100$, $\alpha = 8$, $\beta = 2$.

Nullhypotesen er at $H_0 : \mu_1 - \mu_2 = 0$ og den alternative hypotese er $H_1 : \mu_1 - \mu_2 \neq 0$. Denne nullhypotese undersøges ved hjælp af en uparret t-test. Udfra stikprøverne udføres der to-sidet uparret t-test, ved hjælp af den indbyggede funktion `t.test`.

```
#t.test ud fra de to stikprøver
t1 <- t.test(Stik1, Stik2, alternative = "two.sided", mu = 0, conf.level = 0.95)
#Det tilhørende konfidensinterval fra t.testen
konfinterval <- t1$conf.int
```

Udfra t-testen fås et konfidensinterval på $[-0.6544852, -0.5919858]$. Med et signifikansniveau på 5%, vil den observerede forskel mellem populationernes middelværdier ligge i dette interval i 95% af tilfældene, ifølge t-testen. Dækningsgraden af dette konfidensinterval kan undersøges ved at trække nye stikprøver fra populationerne, i alt 10,000 gange. Dette gøres ved at udregne den nye observerede forskel i middelværdi og sammenligne med konfidensintervallet. Hvis den observerede forskel er indeholdt i konfidensintervallet er det en succes, ellers er det en fiasko.

```
#Undersøg dækningsgraden af konfidensintervallet ved udtræk af nye stikprøver.
res <- replicate(10000, {
  x1 <- sample(pop_hs, 100)
  x2 <- sample(pop_vs, 100)

  mean_diff <- mean(x1) - mean(x2)

  konfinterval[1L] <= mean_diff & konfinterval[2L] >= mean_diff
})
tf <- table(res)
```

Figure 4.6: Andel af succeser og fiaskoer

Det fremgår at i 31.43% af tilfældene ligger den observerede forskel i middelværdierne uden for konfidensintervallet. Dette stemmer ikke overens med antagelsen om at 5% burde ligge udenfor. Konklusionen er derfor at en t-test udført på ikke-normalfordelte stikprøver ikke nødvendigvis giver et retvisende resultat.

Chapter 5

Resampling-metoder

FIXME: Bland med kapitel 3 (overvej)

I de følgende to underafsnit beskrives to metoder, som anvendes til at permutere eller resample fra en stikprøve.

5.1 Permutationstest

For at en t-test overholder den nominelle risiko for fejl, er der visse antagelser, der skal være opfyldt, heriblandt, at stikprøven skal være normalfordelt og opnået ved tilfældig stikprøveudtagning. Er disse antagelser ikke opfyldt, giver testen ikke et retvisende resultat, hvilket i dette afsnit vil eftervises ved hjælp af en permutationstest i R. Først gives en kort introduktion til permutationer, herefter en forklaring af, hvad en permutationstest er, og til sidst modbevises de resultater, opnåes, hvis en t-test udføres på data, der ikke overholder kravene.

5.1.1 Permutationer

I dette afsnit defineres permutationer, og der gives et kort eksempel på en permutation af en ordnet liste af tre elementer. Afsnittet er skrevet på baggrund af (?).

Givet en ordnet liste, $X = [x_1, x_2, \dots, x_n]$, er en permutation, π , en omarrangering af listens elementer, $X_\pi = [x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(n)}]$. Antallet af mulige permutationer af en ordnet liste, X , af længde n , er givet ved $n!$.

Som eksempel, er en mulig permutation, π_1 , af $A = [1, 2, 3]$ givet ved $A_{\pi_1} = [2, 1, 3]$, og der er i alt $|A| = 3! = 1 \cdot 2 \cdot 3 = 6$ mulige permutationer af listen.

5.1.2 Test ved hjælp af permutationer

I dette afsnit gives en forklaring af, hvad en permutationstest er. Afsnittet er skrevet på baggrund af (?).

Antag, at en tilfældig stikprøveudtagning af en population kan udtrykkes ved $X = [x_1, x_2, \dots, x_n]$. Der opstilles en nulhypotese H_0 og en alternativ hypotese, H_1 , der er relevante at undersøge. Der kan så vælges en teststørrelse at undersøge for at teste validiteten af H_0 . Værdien af denne teststørrelse er givet ved S_{obs} for stikprøven.

En permutationstest går ud på at forsøge at måle evidens imod H_0 , som i en almindelig t-test. Dette gøres ved at beregne teststørrelsen for samtlige permutationer af X . Denne mængde af permutationer er givet ved $\Pi_X = \{X_{\pi(1)}, X_{\pi(2)}, \dots, X_{\pi(m)}\}$, hvor m er antallet af mulige permutationer af X . Teststørrelsen for den i 'te permutation benævnes S_{π_i} .

Ved nu at undersøge, hvor stor en andel af S_{π_i} , der har en mere ekstrem værdi end S_{obs} , findes en p -værdi for testen. Dette kan skrives som $p = \frac{\#S_{\pi_i} \text{ mere ekstrem end } S_{obs}}{m}$.

Denne form for test har den fordel, at den antager meget lidt omkring stikprøven. I særdeleshed antager den hverken tilfældig stikprøveudtagning, eller at populationen, fra hvilken stikprøven stammer, er normalfordelt. Der er dog visse ulemper ved en permutationstest. For det første, skal det antages, at den del af data, der permuteres, er ombyttelig. For det andet er det ikke alle teststørrelser, der kan testes ved hjælp af en permutationstest - for eksempel vil middelværdien af samtlige permutationer være lig hinanden. For det tredje bliver antallet af mulige permutationer hurtigt meget stor, når størrelsen af stikprøven stiger - hvis der for eksempel er $n = 10$ datapunkter i stikprøven, vil der være $10! = 3,628,800$ mulige permutationer af stikprøven.

Netop på grund af ombytteligheden af stikprøven, fungerer permutationstests. Hvis stikprøven er ombyttelig, vil hver eneste permutation af stikprøven, inklusiv stikprøven selv, være lige sandsynlige. Dette medfører, at hvis H_0 er sand, vil enhver difference imellem S_{obs} og S_{π_i} være lille og tilfældig. Hvis værdien for S_{obs} på den måde konkluderes at være væsentlig anderledes, tyder det på, at H_0 skal forkastes.

Det store antal af permutationer afhjælpes ved kun at beregne S_{π_i} for et passende antal, tilfældigt udvalgte, permutationer af stikprøven, og et acceptabelt resultat vil stadig blive opnået.

Eksempel

FIXME: (Lav et eksempel, der tager udgangspunkt i realistisk data. Overvej, om eksemplet skal slettes.)

Antag, at det ønskes undersøgt, om middelværdien af to populationer, T_1 og T_2 , er signifikant anderledes. Til dette er nedenstående stikprøver tilgængelige.

```
t_1 <- c(70, 75)
t_2 <- c(76, 72)
mean_difference <- abs(mean(t_1) - mean(t_2))
mean_difference
```

```
## [1] 1.5
```

Der opstilles en nulhypotese, at middelværdien af t_1 , \bar{t}_1 er lig middelværdien af t_2 , \bar{t}_2 - $H_0 : \bar{t}_1 - \bar{t}_2 = 0$, med den alternative hypotese $H_1 : \bar{t}_1 - \bar{t}_2 \neq 0$. Teststørrelsen vil i dette tilfælde være forskellen mellem t_1 og t_2 , $S_{obs} = |t_1 - t_2|$. Hvorvidt denne forskel er signifikant kan nu undersøges ved at beregne forskellen i middelværdi for samtlige permutationer af stikprøven.

I nedenstående kode oprettes en matrix, hvori hver søjle repræsenterer en permutation af stikprøven, hvor de to første rækker er værdier for t_1 og de to sidste rækker er værdier for t_2 .

```
#set.seed(29)
t <- c(70, 75, 76, 72)
perm_samples <- matrix(0, nrow = 4, ncol = 6)
for(i in 1:6){
  perm_samples[,i] <- sample(t, size = 4, replace = FALSE)
}
perm_samples
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]   75   70   75   70   76   75
## [2,]   76   75   70   75   70   70
## [3,]   72   76   72   72   72   76
## [4,]   70   72   76   76   75   72
```

I nedenstående kode beregnes den absolutte forskel i middelværdierne af hver af de permuterede datasæt, og gemmer den forskel i en liste.

```
mean_list <- c()
for(i in 1:6){
  mean_list[i] <- abs(mean(perm_samples[1:2, i]) - mean(perm_samples[3:4, i]))
}
mean_list
```

```
## [1] 4.5 1.5 1.5 1.5 0.5 1.5
```

Nu kan p-værdien beregnes, hvilket gøres i nedenstående kodestykke.

```
extreme_values <- 0
for(i in 1:length(mean_list)){
  if(mean_list[i] >= mean_difference){
    extreme_values <- extreme_values + 1
  }
}
```

```

}
p_value <- extreme_values/length(mean_list)
p_value

```

```
## [1] 0.8333333
```

Der er altså ikke belæg for at forkaste nulhypotesen.

5.2 Bootstrap

I det følgende afsnit vil den teoretiske del af en resampling-metode, kaldet bootstrap, blive beskrevet. Senere vil det blive undersøgt, hvordan bootstrap kan bruges praktisk.

Der findes forskellige bootstrap-metoder, som varierer på forskellige punkter. Valget af bootstrap-metode afhænger af den individuelle situation, hvor der skal udføres statistisk inferens. Der gøres opmærksom på, at i den resterende del af rapporten, vil ordet bootstrap henvise til den ikke-parametriske bootstrap-metode. Ikke-parametrisk bootstrap, er når der ikke sættes specifikke antagelser eller en præcis model for populationen, når undersøgelsen udføres. Derimod antages det, at en stikprøve er repræsentativ for hele populationen, (? , side 3).

Bootstrap er en resampling-metode, der bruges til at generere yderligere datasæt ud fra den givne stikprøve, hvor målet er at udføre statistisk inferens for en valgt teststørrelse. For eksempel kan bootstrap give et indblik i tendenser for teststørrelsen, såsom standardfejlen og forventningsræthed, eller udregne konfidensintervaller. Der gøres opmærksom på, at bootstrap ikke kan bruges til at få et bedre estimat for parameteren, da bootstrap-fordelingen er centreret omkring stikprøvens estimat, se Figur ??, for eksempel middelværdien $\hat{\mu}$, og ikke populationens middelværdi, μ , (? , s. 114).

Figure 5.1: (a) Populationens fordeling, $N(23, 49)$. (b) Fordelingen på en stikprøve af størrelsen 50. (c) Den teoretiske fordeling af stikprøvens middelværdi. (d) Fordelingen af bootstrap-stikprøvers middelværdi. De stiplede linjer repræsenterer middelværdien, [MathStat, side 108].

Hver bootstrap-stikprøve har størrelsen n , altså den samme størrelse som stikprøven. Bootstrap opererer med tilbagelægning, så der er en sandsynlighed for, at et givent datapunkt bliver udtaget mere end en gang. Samtidig er der en sandsynlighed for, at et datapunkt slet ikke bliver udvalgt. Det er relevant at undersøge, hvor mange af de oprindelige observationer, som i gennemsnit medtages i nye bootstrap-stikprøver, og ligeledes, hvor mange, som udelades.

Sandsynligheden for, at en specifik observation ikke udtages fra de oprindelige n observationer, er $1 - 1/n$, og sandsynligheden for, at denne observation ikke udtages n gange er $(1 - 1/n)^n$. Når stikprøvestørrelsen, n , går mod uendeligt

gælder, at $(1 - 1/n)^n = 1/e \approx 0.368$. Derfor vil en bootstrap-stikprøve af tilpas stor størrelse indeholde $\approx 63.2\%$ observationer fra den oprindelige stikprøve, og udelade $\approx 36.8\%$, (?).

I alt bliver der genereret B bootstrap-stikprøver, som der hver især udføres statistisk inferens på. Med den computerkraft der er tilgængelig i dag, anbefales der af kilden, (?), mindst 10,000 resamples, derved $B \geq 10,000$, for at få et nøjagtigt estimat. Grunden til, at der ikke genereres et endnu større antal bootstrap-stikprøver end de 10,000 er, at bootstrap-stikprøven generes ud fra den observerede data. Et større B vil derfor ikke medføre yderligere information om populationen, men vil dog medvirke til et mere præcist estimat, (?, 10:20).

Fordelen ved bootstrap er, at selvom der kun er én tilgængelig stikprøve fra den underliggende population, er der stadig mulighed for at estimere stikprøvefordelingen, uden at der kræves yderligere stikprøver fra populationen. Dette skyldes netop antagelsen om, at stikprøven skal være repræsentativ for populationen.

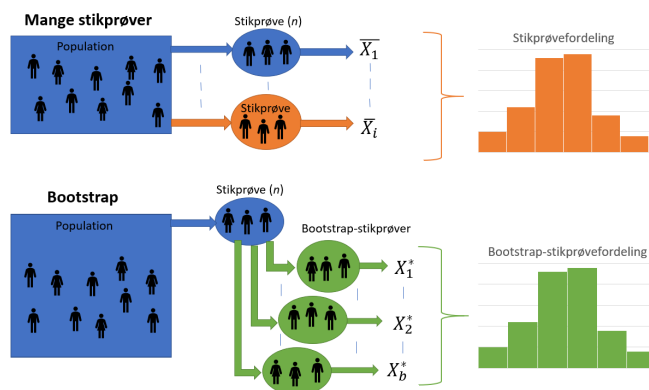


Figure 5.2: Her er illustreret forskellen mellem at finde den teoretiske stikprøvefordeling ved hjælp af mange stikprøver fra populationen (orange), og måden hvorpå stikprøvefordelingen kan findes ved hjælp af kun én stikprøve, der udføres bootstrap på (grøn).

FIXME: (Formler skal kigges igennem, om der er overensstemmelse.)

Der er to hovedårsager til at benytte bootstrap, som beskrevet i (?). For det første, hvis stikprøven ikke er stor, og stikprøvefordelingen derfor heller ikke kan antages at være normalfordelt. For det andet, hvis metoden til at beregne teststørrelsens standardfejl er teoretisk avanceret. Eksempelvis er standardfejlen for middelværdien nem at løse, $\hat{se}(\hat{\mu}) = \frac{S}{\sqrt{n}}$, mens det ikke er tilfældet, hvis det i stedet er afstanden mellem to percentiler, der estimeres.

Chapter 6

Bootstrap anvendelser

FIXME Metaafsnit

6.1 Bootstrap-standardfejl

Der vil i dette afsnit beskrives, hvordan standardfejl af en bootstrap-stikprøve udregnes. I det efterfølgende afsnit vil standardfejlen inddrages i beregningen af konfidensintervallet for en bootstrap-stikprøve. Det følgende afsnit er primært skrevet på baggrund af (?).

Standardafvigelsen for en estimator beskrives som estimatorens standardfejl. Standardfejlen er et udtryk for, hvor stor en afvigelse der er fra populationens parameter til stikprøvens estimat. Jo mindre standardfejlen er, desto mindre er afvigelsen mellem estimatet og parameteren. Som udgangspunkt vil en stikprøves estimat aldrig være lig populationens parameter, fordi der ved udtagning af en stikprøve, i hvert tilfælde vil være variabilitet. Et mål for denne variabilitet er standardfejlen.

Som eksempel vil standardfejlen for estimatet af middelværdien, $\hat{\mu}$, være $se(\hat{\mu}) = \frac{\sigma}{\sqrt{n}}$.

Når der arbejdes med data udover det teoretiske, vil standardafvigelsen for populationen, σ , altid være ukendt. Derfor bruges stikprøvens estimat for standardafvigelsen, S til at beregne den estimerede standardfejl, \hat{se} .

Som eksempel vil den estimerede standardfejl for estimatet af middelværdien, $\hat{\mu}$, være $\hat{se}(\hat{\mu}) = \frac{S}{\sqrt{n}}$, hvor $S = \sqrt{\sum_{i=1}^n \frac{(x_i - \hat{\mu})^2}{n-1}}$ er stikprøvens standardafvigelse for middelværdien, og n er størrelsen på stikprøven.

Såfremt en estimator er normalfordelt eller tilnærmelsesvist er normalfordelt, kan det forventes, at et estimat vil være mindre end én standardfejl fra det forventede i 68% af tilfældene og mindre end to standardfejl fra i 95% af tilfældene.

Dog er dette ikke altid ligetil i virkeligheden, oftest er der ikke tilstrækkelig informationer om populationen eller fordelingen af denne. Samtidig kræver det, at der er nogle specifikke krav som er opfyldt. Disse problemer kan undgås ved at benytte bootstrap til at estimere standardfejlen, givet ved nedstående formel.

$$se(\hat{\theta}) = \sqrt{\frac{1}{B-1} \sum_{b=1}^B (\hat{\theta}_b^* - \bar{\theta})^2}$$

Hvor $\hat{\theta}$ er stikprøvens estimat for den ønskede parameter, B er antal bootstrap-stikprøver, $\hat{\theta}_b^*$ er estimatet for den b 'te bootstrap-stikprøve og $\bar{\theta} = (\frac{1}{B}) \sum_{b=1}^B \hat{\theta}_b^*$, (?).

FIXME overgang til KI Anvendelse af standardfejl i forbindelse med bootstrap-stikprøver...

6.2 Bootstrap-konfidensintervaller

FIXME: (Bedre begrundelse for, hvorfor vi har flere forskellige metoder på konfidensintervaller med. Er der nogle, der er bedre end andre?)

I dette afsnit beskrives, hvorledes konfidensintervallet for en bootstrap-stikprøves beregnes ved hjælp af standardfejlen, percentiler, basic-metoden og t-metoden.

Et 95% konfidensinterval på en normalfordelt stikprøve kan udregnes ved $KI = [\hat{\theta} - 1.96 \cdot se(\hat{\theta}), \hat{\theta} + 1.96 \cdot se(\hat{\theta})]$, (?). Hvis ikke fordelingen er kendt, er det ikke muligt at udregne et konfidensinterval således. Her er det i stedet muligt at benytte bootstrap til at udregne et konfidensinterval, hvilket kan gøres ved hjælp af forskellige metoder.

6.2.1 Percentilmetoden

En intuitiv fremgangsmåde til at bestemme et konfidensinterval ved percentiler, er at bruge det $(B(\frac{\alpha}{2}))$ 'te og $(B(1 - \frac{\alpha}{2}))$ 'te percentil, hvor B er antallet af bootstrap-repetitioner. Lad $\Theta^* = [\vartheta_1, \dots, \vartheta_B]$ være en sorteret liste af bootstrap-estimer. Derved er formelen for et konfidensinterval på baggrund af percentiler:

$$KI_p = [\theta_{(\alpha/2)}^{\sim}, \theta_{(1-\alpha/2)}^{\sim}]$$

Hvor θ_i^{\sim} er det i 'te percentil i Θ^* . (?)

Eksempel

I kodestykket nedenfor udtages en tilfældig stikprøve fra en standard normalfordeling. Herefter laves 10,000 bootstrap-stikprøver, som middelværdien udregnes på. Til sidst sorteres middelværdierne i en liste.

```
data <- rnorm(100, mean = 0, sd = 1) # Tilfældig stikprøve

n <- length(data)
B <- 10000 # Bootstrap-repetitioner

bootstrap_fordeling <- replicate(B, {
  x <- mean(sample(data, size = n, replace = TRUE))
}) # Bootstrap over middelværdien

SortedData <- sort(bootstrap_fordeling) # Bootstrap-fordelinger sorteret
```

Dernæst vælges et konfidensniveau, som regel 90%, 95% eller 99%.

Bootstrap-konfidensinterval ved hjælp af percentiler kan findes ved:

```
KI_niveau <- 0.95 # Konfidensinterval
alpha <- 1-KI_niveau # Benyttes til percentilerne

IndexLower <- round(B * alpha/2) # Nedre percentil
IndexUpper <- round(B * (1-alpha/2)) # Øvre percentil

KI_Percentil <- c(Lower = SortedData[IndexLower], Upper = SortedData[IndexUpper])
# Konfidensinterval vha. percentiler

KI_Percentil

##      Lower      Upper
## -0.2471011  0.1543761
```

FIXME Mangler kilde, plus hvis eventuelt hvordan fordelingen ser ud hvis den er skæv.

Ulempen ved brug af percentiler er, at konfidensintervallet ofte er ukorrekt hvis fordelingen af stikprøven er skæv. En mere præcis metode er basic-metoden.

6.2.2 Basic-metoden

Denne type konfidensinterval ud fra bootstrap er også kendt som *reversed percentile interval*. Denne metode benytter formelen:

$$KI_b = \left[2\hat{\theta} - \theta_{(1-\alpha/2)}^{\sim}, 2\hat{\theta} - \theta_{(\alpha/2)}^{\sim} \right]$$

Hvor θ_i^{\sim} er det i 'te percentil i Θ^* og $\hat{\theta}$ er middelværdien af stikprøven, (?).

Eksempel

I kodelykket nedenfor beregnes middelværdien af stikprøven og benyttes i formelen.

```

theta_hat <- mean(data)

KI_Basic <- c(Lower = 2*theta_hat-SortedData[IndexUpper],
              Upper = 2*theta_hat-SortedData[IndexLower])

KI_Basic

##      Lower      Upper
## -0.2375702  0.1639069

```

6.2.3 T-metoden

Konfidensintervaller kan også findes ved hjælp af standardfejl fra stikprøven, hvis stikprøven er tilnærmelsesvist t-fordelt.

Formlen for konfidensintervallet er:

$$KI_t = \left[\hat{\theta} - t_{(1-\alpha/2)}^* \cdot \hat{se}(\theta), \hat{\theta} - t_{(\alpha/2)}^* \cdot \hat{se}(\theta) \right]$$

Hvor $\hat{\theta}$ er middelværdien af stikprøven, $\hat{\vartheta}$ er middelværdien for Θ^* og $t^* = \frac{\hat{\vartheta} - \hat{\theta}}{\hat{se}(\hat{\vartheta})}$, (?).

Eksempel

I kodelykket forneden beregnes t^* for det nedre og øvre percentil. Dernæst beregnes konfidensintervallet ud fra formelen.

```

tLower <- (SortedData[IndexUpper]-theta_hat) /
  (sd(bootstrap_fordeling)) # SD udregner standardfejl
tUpper <- (SortedData[IndexLower]-theta_hat) /
  (sd(bootstrap_fordeling)) # SD udregner standardfejl

KI_Normal <- c(Lower = theta_hat - tLower * (sd(data)/sqrt(n)),
              Upper = theta_hat - tUpper * (sd(data)/sqrt(n)))

KI_Normal

##      Lower      Upper
## -0.2394553  0.1658837

```

FIXME: (Eventuel sammenligning / Undersøgelse af dækningsgraden - Tænk det gøres i dataanalysen)

6.3 Bootstrap-hypotesetest

Som nævnt i afsnit ??, skal følgende tre antagelser være opfyldt, for at garantere korrektheden af resultaterne af en t-test.

1. Variablen er kvantitativ.
2. Stikprøveudtagning er udført med tilfældighed.
3. Populationen er normalfordelt.

Er antagelserne ikke opfyldt kan bootstrap anvendes til at udføre t-test, og i så fald kaldes det i det følgende for en bootstrap-test. I følgende to afsnit gennemgås først fremgangsmåden for en parret bootstrap-test, og dernæst fremgangsmåden for en uparret bootstrap-test.

6.3.1 Parret bootstrap-test

Lad to parrede stikprøver være givet, $X = [x_1, x_2, \dots, x_n]$ og $Y = [y_1, y_2, \dots, y_n]$. Der oprettes et tredje datasæt, Z , som består af differencerne mellem x_i og y_i , $Z = [x_1 - y_1, x_2 - y_2, \dots, x_n - y_n]$. Ved hjælp af det nye datasæt er det muligt at udregne teststørrelsen, $t_{obs} = \hat{Z}$.

Så opstilles der en nulhypotese, $H_0 : \mu = 0$, hvor μ angiver den sande middelværdi for differencerne for X og Y , og en alternativ hypotese, $H_1 : \mu \neq 0$, samt et signifikansniveau, $\alpha = 0.05$.

Først forenes de to stikprøver til en samlet stikprøve med størrelsen $2n$ observationer. Derefter laves en bootstrap-stikprøve af $2n$ observationer med tilbagelægning fra den samlede stikprøve. Herefter trækkes den første halvdel af indgangene i bootstrap-stikprøven, som betegnes X^* , og den sidste halvdel, der betegnes Y^* . Lad $X_i^* = [x_{1,i}^*, x_{2,i}^*, \dots, x_{n,i}^*]$ betegne den i 'te bootstrap-stikprøve for X og tilsvarende for Y , (? , slide 2).

På baggrund af bootstrap-stikprøverne for X og Y , kan der nu udregnes B nye teststørrelser, $t_i^* = \hat{Z}_i^*$, hvor $Z_i^* = [x_{1,i}^* - y_{1,i}^*, x_{2,i}^* - y_{2,i}^*, \dots, x_{n,i}^* - y_{n,i}^*]$, (?).

Herefter kan p-værdien udregnes, som antal gange bootstrap-teststørrelsen er mere ekstrem end den observerede teststørrelse. Dette gøres ved at finde den mindste af de to ensidede p-værdier og gange den med 2, (?).

$$p_{mindre} = \frac{\text{antal gange } \{t^* < t_{obs}\}}{B} \quad (6.1)$$

$$p_{større} = \frac{\text{antal gange } \{t^* > t_{obs}\}}{B} \quad (6.2)$$

$$p_{tosidet} = 2 \cdot \min(p_{mindre}, p_{større}) \quad (6.3)$$

Forkast H_0 , hvis $p_{tosidet} < \alpha$.

Eksempel

I nedenstående kode, vises et eksempel på en parret bootstrap-test.

```
# Opretter stikprøver
n <- 15
```

```

stik1 <- sample(pop_vs, size = n)
stik2 <- sample(pop_hs, size = n)

# Beregner t_obs
stikdiff <- NULL
for(i in 1:n){
  stikdiff[i] <- abs(stik1[i] - stik2[i])
}
t_obs <- mean(stikdiff)

# Opretter bootstrap-stikprøver
B <- 10000
boot <- replicate(B, sample(x = c(stik1, stik2), size = 2*n, replace = TRUE))

boot1 <- boot[1:15,]
boot2 <- boot[16:30,]

bootdiff <- matrix(data = boot1, nrow = n, ncol = B/2)
bootmeans <- NULL
for(i in 1:B/2){
  bootdiff[1:15, i] <- abs(boot1[1:15, i] - boot2[1:15, i])
  bootmeans[i] <- mean(bootdiff[1:15, i])
}

andel_mindre <- bootmeans < t_obs

p_stoerre <- (sum(andel_mindre)) / length(andel_mindre)
p_mindre <- (length(andel_mindre)-(sum(andel_mindre))) / length(andel_mindre)

# P-værdien for en tosidet bootstrap-test
p_tosidet <- 2*min(p_mindre, p_stoerre)
p_tosidet

## [1] 0.0012

```

Der ses altså at p-værdien er lig 0.0012, hvilket er mindre end signifikansniveauet. Her vil nulhypotesen forkastes, da der er evidens for at differencen ikke er 0.

6.3.2 Uparret bootstrap-test

Lad to uafhængige uparrede stikprøver, $X = [x_1, x_2, \dots, x_n]$ og $Y = [y_1, y_2, \dots, y_m]$, med ens varians, være givet.

På baggrund af forskellen i de to stikprøvers middelværdi, er det muligt at udregne en teststørrelse, $t_{obs} = \hat{X} - \hat{Y}$.

Så opstilles der en nulhypotese, $H_0 : \mu_X = \mu_Y$, hvor μ_X og μ_Y er de sande

middelværdier for populationerne, hvorfra stikprøverne blev udtrukket og en alternativ hypotese, $H_1 : \mu_x \neq \mu_y$, samt et signifikansniveau, $\alpha = 0.05$.

Først forenes de to stikprøver til en samlet stikprøve med størrelsen $n + m$ observationer. Derefter laves en bootstrap-stikprøve af $n + m$ observationer med tilbagelægning fra den samlede stikprøve. Herefter udregnes middelværdien af de første n observationer, som kaldes \hat{X}_1^* . Desuden udregnes middelværdien af de resterende m observationer, der kaldes \hat{Y}_1^* . Til sidst udregnes bootstrap-teststørrelsen $t_1^* = \hat{X}_1^* - \hat{Y}_1^*$, (?).

Trin to til fem i ovenstående gentages i alt B gange, hvilket giver B teststørrelser.

Herefter kan p-værdien udregnes, som antal gange bootstrap-teststørrelsen er mere ekstrem end den observerede teststørrelse. Dette gøres ved at finde den mindste af de to ensidede p-værdier og gange den med 2, (?).

$$p_{mindre} = \frac{\text{antal gange } \{t^* < t_{obs}\}}{B} \quad (6.4)$$

$$p_{større} = \frac{\text{antal gange } \{t^* > t_{obs}\}}{B} \quad (6.5)$$

$$p_{tosidet} = 2 \cdot \min(p_{mindre}, p_{større}) \quad (6.6)$$

Forkast H_0 , hvis $p_{tosidet} < \alpha$.

Eksempel

Nu vil der vises et eksempel på en uparret bootstrap t-test.

```
# Opretter to stikprøver
n2 = 15
m2 = 30
stik3 <- sample(pop_vs, size = n2)
stik4 <- sample(pop_hs, size = m2)

# Beregner t_obs
t_obs2 <- mean(stik3) - mean(stik4)

#Opretter bootstrap-stikprøver og returnerer difference i middelværdi
B2 <- 10000

res <- replicate(B2, {

  boot <- sample(c(stik3, stik4), replace = TRUE)

  bootx <- boot[1 : n2]
  booty <- boot[(n2+1) : (n2+m2)]
```

```
diffmean <- (mean(bootx) - mean(booty))
diffmean

})

andel_mindre2 <- res < t_obs2

p_stoerre2 <- (sum(andel_mindre2)) / length(andel_mindre2)
p_mindre2 <- (length(andel_mindre2)-(sum(andel_mindre2))) / length(andel_mindre2)

# P-værdien for en tosidet bootstrap-test
p_tosidet2 <- 2*min(p_mindre2, p_stoerre2)
p_tosidet2

## [1] 0
```

Der ses altså at p-værdien er lig 0, hvilket er mindre end signifikansniveauet. Her vil nulhypotesen forkastes, da der er evidens for at differencen ikke er 0.

Chapter 7

Diskussion

Diskutér

Chapter 8

Konklusion

Konkludér

Chapter 9

Perspektivering

Perspektivér