Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica
Delft Institute of Applied Mathematics

Een analyse van de bootstrap binnen eindige populaties

(Engelse titel: An analysis of the bootstrap within finite populations))

Verslag ten behoeve van het
Delft Institute of Applied Mathematics
als onderdeel ter verkrijging

van de graad van

BACHELOR OF SCIENCE
in
TECHNISCHE WISKUNDE

door

KASPER KOOIJMAN

Delft, Nederland
Augustus 2016

BSc verslag TECHNISCHE WISKUNDE

"Een analyse van de bootstrap binnen eindige populaties"

(Engelse titel: "An analysis of the bootstrap within finite populations)"

KASPER KOOIJMAN

Technische Universiteit Delft

Begeleider

Dr. G.F. Nane

Overige commissieleden

Drs. E.M. van Elderen

Dr. J.L.A. Dubbeldam

Dr. L. Waltman

Augustus, 2016

Delft

Inhoudsopgave

1 Introductie

Binnen de citation analysis en de toegepaste statistiek is de bootstrapmethode erg populair. Onderzoekers maken veelal gebruik van programma's als *R, SPSS* en *Stata* om met behulp van de bootstrap conclusies te trekken over een populatie, gegeven een sample. De meeste standaard statistische bootstrapmethoden, en dus de populaire statistische programma's, gaan uit van een oneindig grote populatie, dit terwijl populaties meestal maar eindig groot zijn binnen de toegepaste statistiek. Bijvoorbeeld in de *citation analysis*, waarin alle publicaties van een universiteit of een tijdschrift eindig zijn. We kijken naar de invloed van deze standaardmethoden op eindige populaties en proberen hier met behulp van het statistisch pakket R betere methodes voor te vinden.

De bootstrap werd in 1979 ontwikkeld door Efron[6] als afleiding van de *Jack knife*-methode. In R is een bootstrapfunctie geschreven, gebaseerd op de methoden van Davison en Hinkley[4]. Hajek[7] publiceerde in 1981 ideeën over sampling binnen een eindige populatie en Sitter[10] en Booth[1] publiceerden respectievelijk in 1992 en 1994 verschillende algoritmes voor de bootstrap binnen een eindige populatie. Deze zijn zeer recentelijk (2016) door Mashregi[9] samen met andere algoritmes overzichtelijk gemaakt.

Nog steeds is er onduidelijkheid over het belang van eindige populatie bootstrapmethoden: wanneer is het belangrijk om onderscheid te maken tussen eindige en oneindige populaties en waarom? We zullen verschillende methodes testen op een bestaande populatie en hier zelf onze conclusies uit trekken.

2 Leiden Ranking

De Leiden Ranking is een rankingsysteem van universiteiten die in 2007 zijn eerste ranking uitbracht [8]. In tegenstelling tot andere rankingsystemen van universiteiten, zoals de Academic Ranking of World Universities (ARWU) en de Times Higher Educations World University Rankings (THE), is de Leiden Ranking exclusief gebaseerd op bibliometrische data verkregen uit de Web of Science (WoS) database [8]. Aan de 500 universiteiten met de meeste publicaties in WoS wordt een plek toegekend in de Leiden Ranking. Het voordeel hiervan is dat de data gebruikt door de Leiden Ranking niet wordt verkregen door deelnemende universiteiten en dus volledig onafhankelijk is.

Waltman noemt ook dat veel rankingsystemen kijken naar een combinatie van verschillende prestatiefactoren. De ARWU neemt bijvoorbeeld ook het aantal alumni van een universiteit dat een nobelprijs heeft gewonnen mee in zijn ranking. De THE kijkt naar een combinatie van lesgeven, onderzoek, citaties, inkomen en 'internationale visie' [8]. Wanneer universiteit A een andere missie heeft dan universiteit B, zal dit van invloed zijn op de score in een ranking. De universiteit die toevallig dezelfde prioriteit heeft als het rankingsysteem zal dan ook veel hoger uitkomen op de lijst. De Leiden Ranking biedt hier echter geen perfecte oplossing voor, wel een verbetering: de Leiden Ranking geeft veel duidelijker aan op welke factor een universiteit goed presteert.

Omdat dit onderzoek vooral gebaseerd is op de manier waarop de Leiden Ranking zijn scores bepaalt en hoe deze zo goed mogelijk benaderd kunnen worden, zal niet verder ingegaan worden op de voor- en nadelen van de Leiden Ranking ten opzichte van de andere universitaire rankings. Wel zal gekeken worden naar de verschillende scores die de Leiden Ranking aan universiteiten toekent en hoe deze verkregen worden.

2.1 Publicatiescores

De citation score (CS) van een publicatie wordt bepaald door het aantal keer dat de publicatie geciteerd wordt. De score van de universiteit wordt vervolgens bepaald door alle scores van publicaties bij elkaar op te tellen. Omdat het aantal publicaties van een universiteit van grote invloed op de CS is, wordt voornamelijk naar de mean citation score (MCS), het gemiddelde aantal citaties, en de mean normalized citation score ($MNCS$) gekeken. De $MNCS$ is het *gemiddelde aantal citaties van de publicaties van een universiteit, genormaliseerd voor verschil in wetenschapsgebied, verschil in jaar van publicatie en verschil in documenttype* [8]. Bij de $MNCS$ moet gedacht worden aan een score van bijvoorbeeld $MNCS = 1.23$. Merk op dat de MCS van een universiteit elk positief getal kan zijn. Dit terwijl de $MNCS$ op zo'n manier wordt berekend dat de $MNCS$ voor alle publicaties in WoS gelijk is aan 1. Als de $MNCS$ van een universiteit groter is dan 1, zoals de TU Delft, dan presteert deze universiteit dus boven het landelijk gemiddelde. Hoe de $MNCS$ precies wordt bepaald is te vinden in in Waltman et al (2012) [8].

Naast de MCS en $MNCS$, wordt ook nog naar de $PP_{top10\%}$ gekeken, dit is de fractie publicaties van een universiteit die tot de 10% meest geciteerde publicaties behoort in het desbetreffende wetenschapsgebied [8]. [1]

Dit zijn de belangrijkste factoren voor dit onderzoek geweest. De Leiden Ranking kijkt ook nog naar samenwerking tussen verschillende universiteiten, internationale samenwerking en afstand tussen samenwerkende universiteiten. Deze factoren zijn voor dit onderzoek buiten

[1] De verschillende wetenschapsgebieden die de Leiden Ranking hanteert zijn: *biomedical en health sciences, life and earth sciences, physical sciences and engineering* en *social sciences and humanities.*

beschouwing gelaten en daar zal hier dus niet verder op ingegaan worden. Een voorbeeld van een deel van de scores ziet er als volgt uit:

Index	Art no	cs	ncs	p_{top10}	Field no
1	207860300155	0	0.00000000	0.00000000	4
2	207867400006	18	9.62790698	1.00000000	1
3	207868500003	4	0.49017774	0.00000000	1
4	207957600006	1	2.17194570	0.01149425	3

Tabel 1: Deel van de scores van de TU Delft in de Leiden Ranking

Merk op dat de NCS van artikel 4 veel hoger is dan die van artikel 3, terwijl deze minder citaties heeft. Dit heeft te maken met het verschil in wetenschapsgebied (field). Een citatie in field 3 (mathematics and computer science) is veel meer waard dan een citatie in field 1 (biomedical and health sciences). De overige gebieden zijn life and earth sciences (2), physical sciences and engineering (4) en social sciences and humanities (5). In sectie 2 zal verder worden ingegaan op deze scores.

Vanaf nu zal deze scriptie een meer theoretische wending krijgen. Er zal gezocht worden naar methodes om van een eindige populatie een zo goed mogelijke schatting van het gemiddelde te maken. We houden vast aan de Leiden Ranking omdat we over de volledige populatie beschikken en de data op dit gebied op interessante wijze verdeeld is. We gebruiken de data daarom om methodes te testen.

3 Data-analyse

We kijken naar de data van de TU Delft van de Leiden Ranking uit periode 1. Dat wil zeggen: de lijst uit het jaar 2011/2012 met artikelen uit de periode 2006-2009, van deze artikelen wordt het aantal citaties tot het jaar 2010 geteld [8]. Er wordt gekeken naar de $MNCS$ en de $PP_{top10\%}$. We beschikken over de scores van 6224 publicaties in deze periode.

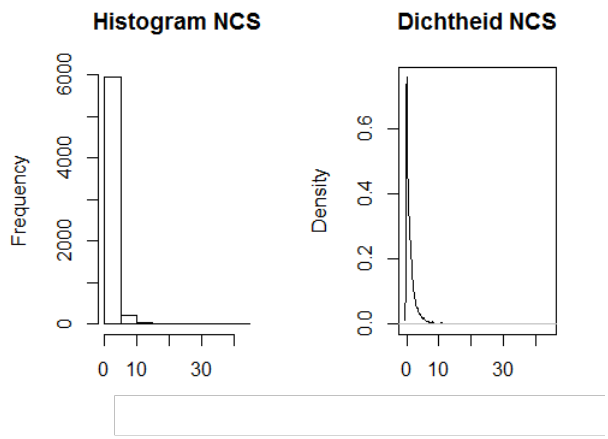
3.1 MNCS

We kijken naar de normalized citation score van publicaties van de TU Delft in periode 1. Een histogram is te vinden in figuur 1a. Het gemiddelde wordt gegeven door $MNCS = 1.275$.

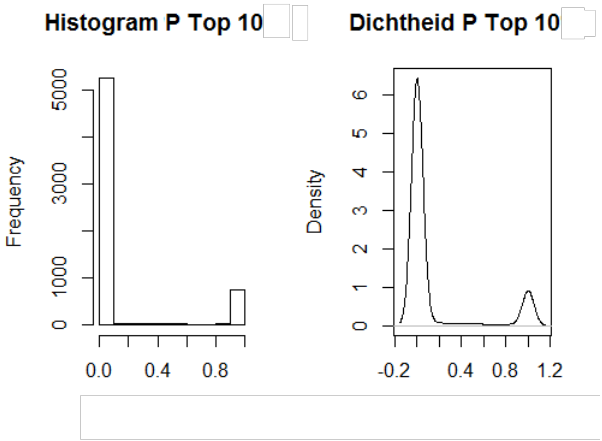
In figuur 1a lijkt onze data erg asymmetrisch verdeeld. Wanneer we de *measure of skewness* berekenen, vinden we:

$$MS = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^3}{\left[\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2\right]^{3/2}} \approx 5.94$$

met $n = 6224$ het aantal publicaties. Een dataset wordt normaal gesproken als symmetrisch beschouwd wanneer $MS \approx 1$. Zoals eerder gezegd is, is de data dus erg asymmetrisch verdeeld.



(a) histogram en verdeling NCS



(b) histogram en verdeling $P_{top10\%}$

3.2 $PP_{Top10\%}$

We kijken naar de $P_{Top10\%}$ van publicaties van de TU Delft in periode 1. Een publicatie krijgt waarde 0 als deze niet in de top 10% staat en waarde 1 als deze er wel in staat. We zien dat een nog groter deel dan van de NCS waarde gelijk aan 0 heeft: namelijk 5180 van de 6224. Een andere piek is te vinden bij 1. De uiteindelijke score van de universiteit, de $PP_{Top10\%}$, wordt gegeven door de proportie vergelijkbare publicaties van een universiteit die in de top 10% meest geciteerde publicaties staan. Oftewel: het gemiddelde. Voor de TU Delft geldt $PP_{Top10\%} = 0.137$.

Naast alle publicaties met waarde 0 en 1 zien we in figuur 1b dat er ook publicaties zijn met waarde $p \in (0, 1)$. Dit heeft te maken met het feit dat er naar vergelijkbare publicates wordt gekeken: *een publicatie met twintig citaties in vier vakgebieden die in één van de vier vakgebieden tot de top* 10% *behoort, krijgt een score van* 0.25 [8]. Daarnaast zijn er ook nog 'gedeelde' plaatsen. Tien publicaties in de $PP_{top10\%}$ met evenveel citaties, zullen allemaal een score 0.1 krijgen.

Merk op dat ook deze data asymmetrisch verdeeld zijn, ditmaal met
$MS \approx 2.13$.

4 Asymmetrie van de data

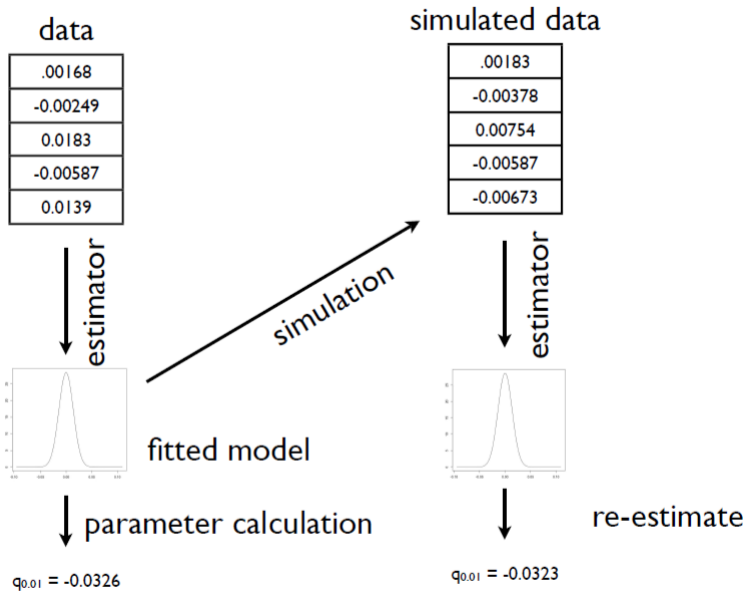
Tijdens het kijken naar de verdeling van de NCS en de $P_{top10\%}$ hebben we veel aandacht besteed aan de asymmetrie van de data. Dit is belangrijk vanwege een gevolg van de centrale limietstelling. De centrale limietstelling zegt grofweg dat het gemiddelde van een groot aantal simple random samples bij benadering een normale verdeling volgt, ongeacht de onderliggende verdeling van de populatie. Voor een normaal verdeelde populatie zal dit al gelden wanneer we kleine samples nemen, echter voor asymmetrische verdelingen is het belangrijk dat de samplegrootte veel groter is om het gemiddelde normaal verdeeld te laten zijn [5]. De samplegrootte is dus van grote invloed op de verdeling van het gemiddelde van de NCS en de $P_{top10\%}$.

5 Bootstrap

De Leiden Ranking maakt gebruik van de bootstrapmethode om zijn stabiliteitsintervallen [2] te bepalen [8]. Dit gegeven, samen met het feit dat deze methode vrij rechtdoorzee en eenvoudig te implementeren is, heeft geleid tot de keuze om deze methode nader te beschouwen. De algemene bootstrapmethode, bedacht door Efron (1979) [6], kent twee varianten: de parametrische en de niet-parametrische.

5.1 Parametrische bootstrap

Gegeven een sample $s = \{y_1, y_2, \ldots, y_n\}$ van populatie $P = \{Y_1, Y_2, \ldots, Y_N\}$, we zijn op zoek naar de parameter θ (in ons geval het gemiddelde). Een parametrische bootstrap probeert aan de hand van de sample s te schatten hoe de populatie verdeeld is, we noemen deze verdelingsfunctie $\hat{F}(x)$ en de bijbehorende parameterschatting $\hat{\theta}$. Vervolgens worden uit $\hat{F}(x)$ nieuwe samples $\hat{x}_1^*, \hat{x}_2^*, \ldots, \hat{x}_B^*$ getrokken, met $B \in \mathbb{N}$ groot [3]. Aan de hand van deze bootstrapsamples wordt vervolgens een nieuwe schatting van $\hat{F}$ gemaakt: $\hat{F}^*$, vervolgens wordt $\hat{\theta}^*$ geschat. Figuur 2, uit [11] geeft een duidelijke schematische weergave.



Figuur 2: parametrische bootstrap

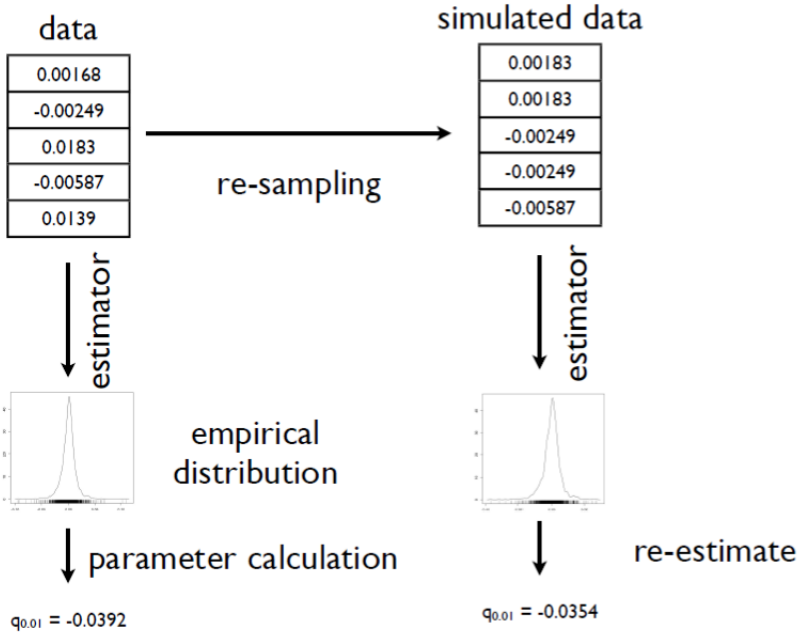
5.2 Niet-parametrische bootstrap

Bij een niet-parametrische bootstrap wordt geen schatting van een verdelingsfunctie $\hat{F}$ gedaan. Een niet-parametrische bootstrap beschouwt de sample $s = \{y_1, \ldots, y_n\}$ als de gehele populatie. Bij een niet-parametrische bootstrap worden B samples $x_1^*, x_2^*, \ldots, x_B^*$ uit de sample s getrokken, met teruglegging, van grootte n. Van iedere sample x_i^* wordt de bootstrapparameter $\hat{\theta}_i^*$ bepaald

[2] Een stabiliteitsinterval geeft de mogelijke waarden van een parameter aan indien de verzameling publicaties van een universiteit verandert. *Een stabiliteitsinterval wordt hetzelfde berekend als een betrouwbaarheidsinterval, maar de interpretatie is anders*[8].

[3] Davison & Hinkley[4] (p. 29) beweren dat $R \geq 1000$ gekozen zou moeten worden voor betrouwbare schattingen.

en met behulp van de verdeling van deze parameters wordt de werkelijke parameter θ geschat. Zie figuur 3 uit [11] voor een schematische weergave hiervan.



Figuur 3: niet-parametrische bootstrap

5.3 Parametrisch of niet-parametrisch?

Zoals gezien in sectie 3, volgt de data van de TU Delft van de Leiden Ranking een vrij ongebruikelijke verdeling. De dichtheid van de NCS zou eventueel een exponentiële verdeling als onderliggende verdeling kunnen hebben, maar voor de $P_{Top10\%}$ blijft het onduidelijk, daarnaast willen we dit probleem zo algemeen mogelijk behandelen, wat ook de keuze voor de niet-parametrische bootstrap bevordert.

6 Bootstrapfunctie R

Het statistisch pakket *R* beschikt over een 'ingebouwde' bootstrapfunctie *boot* en een functie *boot.ci* waarmee betrouwbaarheidsintervallen (BI) berekend worden. *boot.ci* is in staat om vijf verschillende betrouwbaarheidsintervallen te berekenen: een *normal, basic, percentile, BCa* en *studentized*. Voor de *studentized* BI moet echter de variantie van de gewenste parameter gegeven worden, iets welke *boot* niet meegeeft. Dit BI kan met deze functie dus niet berekend worden[4].

De berekening van de betrouwbaarheidsintervallen is gebaseerd op de methodes van Davison en Hinkley [4] en worden uitgebreid door ze uitgelegd. Ook wij zullen het hier nog even bondig bespreken.

6.1 Betrouwbaarheidsintervallen

We bespreken de verschillende methodes om betrouwbaarheidsintervallen te creëren. We gaan uit van een sample s en zijn op zoek naar de parameter θ. Zij t de sample-parameter en

[4]Opmerkelijk is dat een normaal BI ook gebruik maakt van de variantie van de bootstrapparameter. *boot* is wel in staat om dit interval te berekenen. Waarom is vooralsnog onduidelijk.

$T^* = \{\hat{\theta}_1^*, \ldots, \hat{\theta}_B^*\}$ de gevonden bootstrapparameters. We definiëren $T'^* = \{t_1^*, t_2^*, \ldots, t_B^*\}$ als de verzameling gesorteerde parameters, oftewel: $t_1^* \leq t_2^* \leq \ldots \leq t_B^*$ en $t_i^* \in T'^* \Leftrightarrow t_i^* \in T^*$.

6.1.1 Normale benadering

Een normale benadering van een BI gaat ervan uit dat de gezochte parameter normaal verdeeld is. Dit kunnen we controleren door een QQ-plot te maken van T'^* tegenover de normale verdeling. Vervolgens wordt het $(1-\alpha)$-BI gegeven door:

$$\left[t - \sqrt{v} z_{1-\alpha/2}, t + \sqrt{v} z_{1-\alpha/2}\right],$$

met $v = var(\theta)$. Omdat v meestal niet gegeven wordt, moet v worden bepaald met behulp van de bootstrapparameters, en wordt een correctie toegepast. We krijgen:

$$\left[t - b_B - \sqrt{v_B} z_{1-\alpha}, t - b_B + \sqrt{v_B} z_{1-\alpha}\right],$$

met:

$$b_B = \bar{T}^* - t, \tag{1}$$

$$\bar{T}^* = \frac{1}{B} \sum_{i=1}^{B} \theta_i^*, \tag{2}$$

$$v_B = \frac{1}{B-1} \sum_{r=1}^{B} (T_r^* - \bar{T}^*)^2. \tag{3}$$

6.1.2 Basic

Een Basic $(1-\alpha)$-BI wordt bepaald met behulp van de $(B+1)(\alpha/2)$ en $(B+1)(1-\alpha/2)$'de kwantielen van $\hat{\theta}_1^*, \ldots, \hat{\theta}_B^*$. Oftewel: in het geval waar we 1000 bootstrapreplicaties hebben berekend, gebruiken we voor een 95%-BI de waardes die op de 25e en 975e plek van de gesorteerde parameters staan.

$$\left[2t - t^*_{(B+1)(1-\alpha/2)}, 2t - t^*_{(B+1)(\alpha/2)}\right].$$

6.1.3 Studentized

Voor een studentized (ook wel student-t) BI wordt de normale benadering gebruikt, echter wordt $N(0,1)$ vervangen door $z^* = \frac{t^* - t}{\sqrt{v^*}}$. De bootstrapwaarden van θ worden gesorteerd en vervolgens worden de $(B+1)\alpha$ en $(B+1)(1-\alpha)$'de kwantielen bepaald:

$$\left[t - \sqrt{v} z^*_{(B+1)(1-\alpha)}, t - \sqrt{v} z^*_{(B+1)\alpha}\right].$$

Ook hier wordt weer een correctie toegepast voor v, aangezien deze meestal niet gegeven is:

$$\left[t - b_B - \sqrt{v_B} z^*_{(B+1)(1-\alpha)}, t - b_B + \sqrt{v_B} z^*_{(B+1)\alpha}\right].$$

v_B en b_B worden hier op dezelfde manier bepaald als in vergelijking (1), (2) en (3).

Merk op dat zowel voor de student-t als voor de basic BI moet gelden dat $(B+1)\alpha \in \mathbf{N}$, indien dit niet het geval is, gebruiken Davison en Hinkley lineaire interpolatie op de *normal quantile scale*: Zij $k = \lfloor (B+1)\alpha \rfloor$, dan

$$t^*_{(B+1)\alpha} = t^*_k + \frac{\Phi^{-1}(\alpha) - \Phi^{-1}(\frac{k}{B+1})}{\Phi^{-1}(\frac{k+1}{B+1}) - \Phi^{-1}\frac{k}{B+1}}(t^*_{k+1} - t^*_k).$$

6.1.4 Percentile

Voor een percentile BI worden, net als voor de basic, de $(B+1)\alpha/2$ en $(B+1)(1-\alpha/2)$'de kwantielen bepaald. Echter wordt hier helemaal niet naar de sample mean gekeken en wordt het BI gegeven door

$$\left[t^*_{((B+1)\alpha/2)}, t^*_{(B+1)(1-\alpha/2)}\right].$$

6.1.5 BCa

De BCa-bootstrap maakt gebruik van een *bias correction factor* w en een *skewness correction factor* a om goede betrouwbaarheidsintervallen voor asymmetrische verdelingen te creëren. We zoeken een parameter θ, we definiëren de sample-parameter $t = \hat{\theta}$ en de onbekende transformatie $h(.)$, zdd $U = h(T)$ de getransformeerde populatie is en $h(\theta) = \phi$. w en a zijn vooralsnog ook onbekend. We volgen de methode van Davison en Hinkley [4] (p.203 - 209). Voor U moet gelden:

$$U \sim N\left[\phi - w\sigma(\phi), \sigma^2(\phi)\right], \quad \sigma(\phi) = 1 + a\phi.$$

Vervolgens worden de grenzen van het BI van ϕ op dezelfde manier berekend als het percentile BI en deze worden teruggetransformeerd naar de θ-schaal met behulp van de bootstrapverdeling van T. In Davison en Hinkley [4] wordt uitgebreid beschreven hoe deze worden bepaald. We vinden:

$$\begin{aligned}
\hat{\phi}_\alpha &= u + \sigma(u)\frac{w + z_\alpha}{z - a(w + z_\alpha}, \\
\Rightarrow \theta_\alpha &= h^{-1}(\hat{\phi}_\alpha), \\
\Rightarrow \hat{\theta}_\alpha &= t^*_{((B+1)\tilde{\alpha})}, \\
\text{met } \tilde{\alpha} &= \Phi\left[w + \frac{w + z_\alpha}{1 - a(w + z_\alpha)}\right].
\end{aligned}$$

Vervolgens vinden we $w = \Phi^{-1}\left[\frac{\#\{t^*_r \leq t\}}{B+1}\right]$ en $a = \frac{1}{6}\frac{\sum l_j^3}{(\sum l_j^2)^{3/2}}$ met $l_j = y_j - \bar{y}$ de *empirical influence values* voor het gemiddelde, gedefinieerd op p 46 - 48 van Davison en Hinkley [4].

6.2 Betrouwbaarheidsintervallen

We creëren met behulp van de *boot*- en *boot.ci*-functie een normal, basic, percentile en BCa 95% BI voor de $MNCS$ en $PP_{Top10\%}$ gebaseerd op een sample van grootte $n = 500$ en $B = 1000$ bootstrapreplicaties. We vinden de volgende resultaten:

MNCS	$\hat{\theta}$	1.1796
	Normal	$[0.992, 1.362]$
	Basic	$[0.980, 1.342]$
	Perc	$[1.017, 1.379]$
	Bca	$[1.022, 1.392]$
$PP_{Top10\%}$	$\hat{\theta}$	0.1493
	Normal	$[0.1193, 0.1797]$
	Basic	$[0.1185, 0.1814]$
	Perc	$[0.1172, 0.1801]$
	Bca	$[0.1202, 0.1826]$

Tabel 2: Schattingen van de MNCS en $PP_{Top10\%}$ met verschillende BI.

Herinner dat de werkelijke waarden $\theta_{MNCS} = 1.275$ en $\theta_{PP_{Top10\%}} = 0.137$ zijn. De schattingen zijn dus niet helemaal perfect, maar de werkelijke waarden zijn wel in elk BI bevat.

6.3 Nauwkeurigheid

Nu bekend is hoe de betrouwbaarheidsintervallen worden berekend en hoe ze eruit zien, zijn we benieuwd naar de nauwkeurigheid van de *boot*- en *boot.ci*-functies. We schrijven een programma waarin we 100 samples nemen van grootte $n = 500$. Op iedere sample passen we de functies toe en creëren we met behulp van 1000 bootstrapreplicaties een 95% BI. Vervolgens controleren we voor elk type BI of het werkelijke gemiddelde θ hierin bevat is en we berekenen de gemiddelde lengte van alle betrouwbaarheidsintervallen. We kijken zowel naar de $MNCS$ als naar de $PP_{top10\%}$ en vinden de volgende resultaten[5]:

	Type B.I.	Gem. Lengte	Nauwkeurigheid
NCS	Basic	0.3819	97
	Norm	0.3794	96
	Perc	0.3819	96
	Bca	0.3920	97
$PP_{Top10\%}$	Basic	0.0587	98
	Norm	0.0584	98
	Perc	0.0587	97
	Bca	0.0590	97

Tabel 3: Nauwkeurigheid van de *boot*- en *boot.ci*-functies voor $n = 500$

We zien dat elk type BI erg nauwkeurig is, aangezien voor elk type betrouwbaarheidsinterval in meer dan 95% van de gevallen het werkelijke gemiddelde bevat is. Dit is echter ook het verwachtte resultaat. Toch zijn we niet helemaal tevreden.

We hebben informatie verzameld over de nauwkeurigheid van de bootstrapfunctie van B voor samples van grootte $n = 500$. De vraag is nu wat er gebeurt wanneer we onze samplegrootte gaan variëren. We kiezen $n = 100$.

[5]Zie 14.1 in de appendix

	Type B.I.	Gem. Lengte	Nauwkeurigheid
NCS	Basic	0.7633	88
	Norm	0.7613	90
	Perc	0.7633	91
	Bca	0.8150	92
$PP_{Top10\%}$	Basic	0.1290	90
	Norm	0.1289	92
	Perc	0.1290	94
	Bca	0.1327	96

Tabel 4: Nauwkeurigheid van de *boot*- en *boot.ci*-functies voor $n = 100$

De nauwkeurigheid is hier flink afgenomen en de lengte van de betrouwbaarheidsintervallen is verdubbeld ten opzichte van samples van grootte $n = 500$. Echter hebben we door $n = 100$ te kiezen een fractie $100/6224 \approx 0.016$ van onze populatie gekozen. Er zal meer onderzoek gedaan moeten worden voordat we echt goede conclusies over de nauwkeurigheid van deze methode kunnen trekken. We kiezen $n = 1000$.

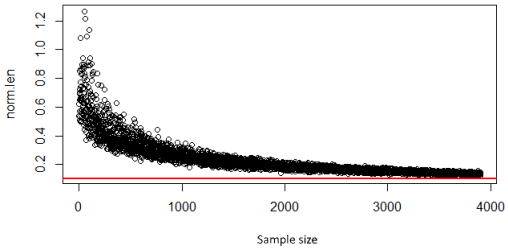
	Type B.I.	Gem. Lengte	Nauwkeurigheid
NCS	Basic	0.2769	96
	Norm	0.2753	96
	Perc	0.2769	96
	Bca	0.2831	97
$PP_{Top10\%}$	Basic	0.0413	96
	Norm	0.0410	98
	Perc	0.0413	98
	Bca	0.0413	99

Tabel 5: Nauwkeurigheid van de *boot*- en *boot.ci*-functies voor $n = 1000$

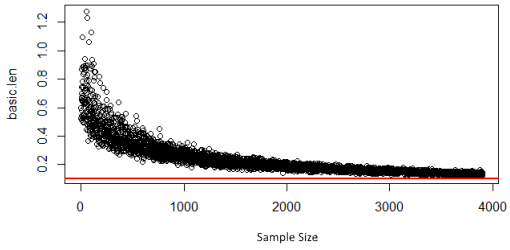
De lengtes van de betrouwbaarheidsintervallen zijn, zoals verwacht, weer afgenomen en de nauwkeurigheid is iets toegenomen ten opzichte van $n = 500$. Wat voornamelijk opvalt, is het verschil in lengte van de betrouwbaarheidsintervallen. Met behulp van 14.2 in de appendix creëren we voor verschillende samples van grootte $n \in \{100, 101, \ldots, 4000\}$ een *basic* en een *normal* BI voor de MNCS en kijken naar de lengtes hiervan. [6] [7]

[6] Om een BCa BI te creëren, dient het aantal bootstrapherhalingen minstens even groot te zijn als de samplegrootte. Voor samples van grootte 2000 en meer is dit een erg inefficiënt en langdurig proces, vandaar dat we dit hier overslaan en alleen naar de basic en normal BI kijken.

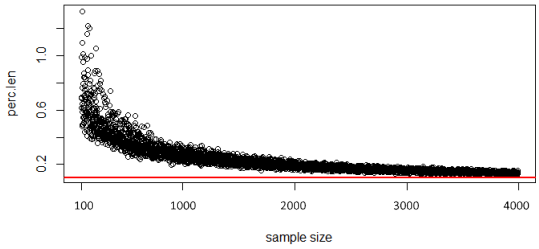
[7] Zowel voor de $PP_{top10\%}$ als voor de MNCS verwachten we hetzelfde gedrag, maar dan met andere waarden. Beide typen BI creëren we voor de MNCS om te kijken of er per type interval ander gedrag optreedt.



(a) Normal



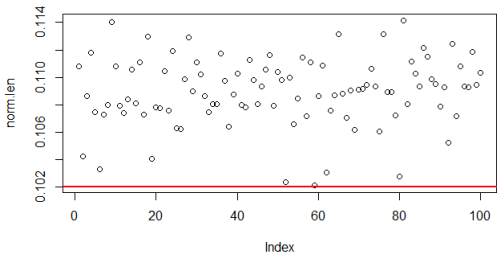
(b) Basic



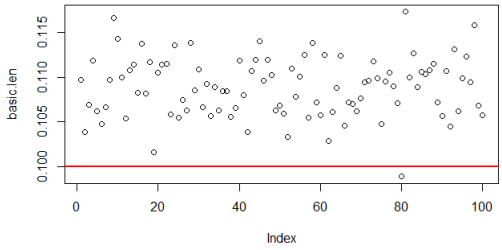
(c) Percentile

Figuur 5: Lengtes verschillende BI voor verschillende n

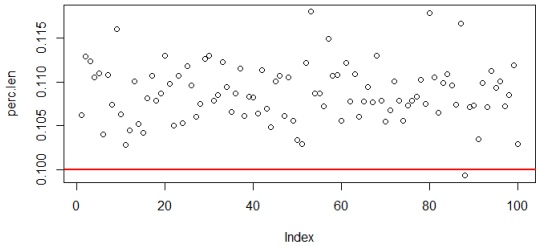
Er lijkt een asymptoot te zijn op een punt hoger dan $y = 0$. Dit gedrag treedt op doordat de *boot* functie van *R* onze populatie als oneindig groot beschouwt. Zelfs als we de hele populatie als sample nemen, verwachten we dat de variantie van de bootstrapparameter nog geen 0 zal zijn [9]. We testen dit door een betrouwbaarheidsinterval te construeren waarvoor we onze sample gelijk aan de populatie kiezen. De exacte waarde van het gemiddelde is dan bekend en $var(\hat{\theta}) = 0$ zou moeten gelden. We gebruiken 14.3 in de appendix om 100 *normal, basic* en *percentile* betrouwbaarheidsintervallen te creëren en vinden de figuren 6a, 6b en 6c.



(a) Normal



(b) Basic



(c) Percentile

Figuur 7: Lengtes 100 verschillende BI met $n = N$

We vinden minimale lengtes van $l_{norm} \approx 0.102$, $l_{basic} \approx 0.09996$ en $l_{perc} = 0.099$. Een echte asymptoot wordt dus niet gevonden, maar het valt wel op dat de *boot.ci* functie veel moeite heeft om betrouwbaarheidsintervallen kleiner dan 0.1 te creëren. Dit terwijl de verwachting nu eigenlijk is dat het gemiddelde exact kan worden gegeven.

7 Eindige populatie correctie

Het feit dat de betrouwbaarheidsintervallen van de bootstrapfuncties van R geen lengte 0 krijgen, wanneer de gehele populatie als sample wordt genomen, komt doordat R geen gebruik maakt van een eindige populatiebootstrap. Oftewel: R gaat er vanuit dat onze populatie oneindig groot is en zal er dus altijd van uitgaan dat we maar een kleine fractie van de hele populatie als sample hebben genomen. De variantie van de bootstrapparameter zal dus nooit naar 0 gaan, zelfs niet als we de hele populatie als sample nemen.

Deze 'fractie' wordt ook wel de *sampling fraction* genoemd en gedefinieerd door $f = \frac{n}{N}$, met $n \in \mathbb{N}$ de sample grootte en $N \in \mathbb{N}$ de populatiegrootte, welke volgens de standaard methoden dus niet eindig is. Vervolgens definiëren we de eindige populatie correctie (*fpc, finite population correction*) door $1 - f$. Stel nu dat de variantie van de bootstrapparameter $\hat{\theta}^*$ gegeven wordt door V^*. Met de fpc wordt deze variantie:

$$V^{*'} = (1 - f)V^*,$$

waaruit volgt dat $V^{*'} \mapsto 0$ als $n \mapsto N$. We bekijken twee bootstrapmethodes die rekening houden met onze eindige populatie.

8 Pseudo population bootstrap

Het idee van de pseudopopulatiebootstrap (PPB) is het creëren van een pseudopopulatie ter grootte van de werkelijke populatie door de gegeven sample te herhalen. Uit de pseudopopulatie worden vervolgens bootstrapsamples getrokken. We gebruiken de algoritmes van Booth et al. (1994) [1] en Chauvet (2007) [3], welke uitgebreid besproken worden door Mashreghi et al. (2016) [9]. We gaan uit van een sample s van grootte n en een populatie P van grootte N (6224 in het geval van publicaties van de TU Delft.)

1. Creëer een Pseudopopulatie P^* van grootte N door s $k = \lfloor \frac{N}{n} \rfloor$ keer te herhalen. P^* wordt afgemaakt door een simple random sample (SRS) zonder teruglegging van grootte $N - nk$ van s te nemen. Merk op dat $nk = N$ als $\frac{N}{n} \in \mathbb{N}$

2. Neem nu een sample s^* van grootte n uit P^* en bepaal θ^*. Herhaal dit B keer, om de bootstrapschatters $\hat{\theta}^*_1, \ldots, \hat{\theta}^*_B$ te bepalen.[8]

3. Om een betrouwbaarheidsinterval te creëren, hebben we een schatting van de variantie nodig:

$$\hat{V}^*_B = \frac{1}{B-1} \sum_{b=1}^{B} (\hat{\theta}^*_b - \hat{\theta}^*_{(.)})^2,$$

   met $\hat{\theta}^*_{(.)} = \frac{1}{B} \sum_{b=1}^{B} \hat{\theta}^*_b$

4. Herhaal bovenstaande 3 stappen D keer om $\hat{V}^*_{1B}, \ldots, \hat{V}^*_{DB}$ te bepalen[9], waarmee we $\hat{V}^*$ kunnen berekenen, gegeven door:

$$\hat{V}^* = \frac{1}{D} \sum_{d=1}^{D} \hat{V}^*_{dB},$$

   het gemiddelde van alle $\hat{V}_{iB}$.

9 Directe bootstrap

In tegenstelling tot de pseudopopulatiebootstrap, is het bij een directe bootstrap niet nodig om de populatie na te bootsen met een pseudopopulatie.

We gebruiken de methode van Sitter (1992) [10], welke uitgebreid bespoken wordt door Mashreghi et al. (2016) [9]. We gaan uit van een sample s van grootte n en populatiegrootte N.

Laat $k' = \left\lfloor \frac{n(1-f'')}{n''(1-f)} \right\rfloor + I_q$, met $I_q \sim Bernoulli(q)$ en $q = \frac{\lfloor k \rfloor^{-1} - k^{-1}}{\lfloor k \rfloor^{-1} - \lceil k \rceil^{-1}}$. Neem $k = \frac{n(1-f'')}{n''(1-f)}$, $f = \frac{n}{N}$ en $f'' = \frac{n''}{n}$.

1. Neem k' SRS zonder teruglegging van grootte $n'' \leq \frac{n}{2-f}$. Voeg al deze samples samen tot één bootstrapsample s^* van grootte $n' = k'n''$. Bepaal vervolgens $\hat{\theta}^*$ uit s^*.

[8] Zoals al eerder genoemd, kiezen we $B = 1000$.

[9] Hier wordt eigenlijk een tweede bootstrap uitegevoerd om de variantie van $\hat{\theta}$ te bepalen. We kiezen daarom ook $D = 1000$.

2. Herhaal stap 1 B keer om $\hat{\theta_1}^*, \ldots, \hat{\theta_B}^*$ te bepalen. En schat de variantie

$$\hat{V}_B^* = \frac{1}{B-1} \sum_{b=1}^{B} (\hat{\theta}_b^* - \hat{\theta}_{(.)}^*)^2,$$

met $\hat{\theta}_{(.)}^* = \frac{1}{B} \sum_{b=1}^{B} \hat{\theta}_b^*$

Een simulatie van verschillende waarden van n'' laat zien dat de directe bootstrap nauwkeuriger resultaten geeft naarmate we n'' groter kiezen, anderzijds duurt elke simulatie ook langer, naarmate n'' groter wordt. Omdat nauwkeurigheid belangrijker is dan snelheid, hebben we voor al onze simulaties n'' steeds vrij dichtbij $\frac{n}{2-f}$ gekozen.

10 Betrouwbaarheidsintervallen

Mashregi [9] beschrijft vervolgens drie verschillende soorten betrouwbaarheidsintervallen: een asymptotisch, een bootstrap-t, en een percentile BI. Deze worden uitgebreid besproken door Mashregi [9] en als volgt gedefinieerd.

10.1 Asymptotisch betrouwbaarheidsinterval

Zij $\hat{V}$ een schatting van de variantie van $\hat{\theta}$ en z_β het β-kwantiel van de standaard normale verdeling. Dan wordt het $1-\alpha$ betrouwbaarheidsinterval gegeven door [9]

$$\left[\hat{\theta} - z_{1-\alpha/2}\sqrt{\hat{V}}, \hat{\theta} - z_{\alpha/2}\sqrt{\hat{V}}\right]. \tag{4}$$

Merk op dat, vanwege symmetrie van de normale verdeling:

$$z_{1-\alpha/2} = -z_{\alpha/2},$$

waardoor interval (4) gelijk is aan het normale BI uit sectie 6.1.1.

10.2 Bootstrap-t betrouwbaarheidsinterval

Een bootstrap-t betrouwbaarheidsinterval, ook wel bekend als student-t, berust op de verdeling van $(\hat{\theta} - \theta)/\sqrt{\hat{V}}$. Zij [9]

$$\hat{K}_{n;*}(x) = P^*\left(\frac{\hat{\theta}^* - \theta^*}{\sqrt{\hat{V}}} \leq x\right),$$

dan definieert Mashregi [9] het bootstrap-t BI door

$$\left[\hat{\theta} - \sqrt{\hat{V}}\hat{K}_{n;*}^{-1}(1-\alpha/2), \hat{\theta} - \sqrt{\hat{V}}\hat{K}_{n;*}^{-1}(\alpha/2)\right].$$

Merk op dat dit interval gelijk is aan het student BI in sectie 6.1.3

10.3 Percentile betrouwbaarheidsinterval

Het percentile interval wordt gedefinieerd door de $B\alpha/2$ en $B(1-\alpha/2)$'de percentielen van de bootstrapschattingen, net als sectie 6.1.4:

$$\left[\hat{\theta}_{B\alpha/2}^*, \hat{\theta}_{B(1-\alpha/2)}^*\right].$$

11 Implementatie in *R*

We hebben de pseudopopulatie- en de directe bootstrap geïmplementeerd met behulp van de codes 14.4 en 14.5 in de appendix. De PPB maakt gebruik van erg veel *for-loops* en is daarom vrij inefficiënt. We hebben dit niet proberen te verbeteren, maar dit zou interessant kunnen zijn voor de toekomst. Het directe algoritme is een stuk sneller, maar hier moeten een aantal waarden zelf gekozen worden. In stap 1 worden k' SRS van grootte $n'' \leq \frac{n}{2-f}$ gekozen. Een simulatie heeft ons laten zien dat de bootstrap nauwkeuriger wordt, naarmate n'' dichter bij $\frac{n}{2-f}$ ligt, echter wordt het algoritme hier ook langzamer van. We hebben niet verder gezocht naar een optimale waarde van n''. Nauwkeurigheid heeft de prioriteit gekregen, wat steeds heeft geleid tot de keuze van n'' dichtbij $\frac{n}{2-f}$. n'' Zo optimaal mogelijk kiezen zou ook een interessant onderzoeksproject voor de toekomst kunnen zijn.

12 Resultaten

We implementeren beide bootstrapmethodes in R en proberen van de NCS en de $PP_{top10\%}$ het gemiddelde te schatten. We nemen samplegrootte $n = 500$, maken 1000 bootstrapiteraties en creëren 95%-betrouwbaarheidsintervallen, naast bovengenoemde intervallen creëren we ook een basic en BCa interval, beschreven in secties 6.1.2 en 6.1.5. Om consistent te blijven, zullen we vanaf nu de asymptotische en bootstrap-t betrouwbaarheidsintervallen respectievelijk normaal en student noemen. Met behulp van 14.4 en 14.5 in de appendix vinden we de volgende resultaten:

Score	Bootstrap	Type B.I.	Gem. Lengte	Nauwkeurigheid	$\frac{length_{boot.ci}}{length_{finite}}$
MNCS	PPB	Basic	0.3617	93	1.0558
		Norm	0.3616	93	1.0492
		Student	0.3617	92	nvt
		Bca	0.3471	93	1.1214
		Perc	0.3617	95	1.1214
	DA	Basic	0.3722	92	1.0261
		Norm	0.3715	95	1.0212
		Student	0.3722	92	nvt
		Bca	0.3576	93	1.0962
		Perc	0.3722	95	1.0261
$PP_{top10\%}$	PPB	Basic	0.05580	95	1.0520
		Norm	0.05588	96	1.0447
		Student	0.05580	95	nvt
		Bca	0.05577	93	1.0424
		Perc	0.05580	96	1.0520
	DA	Basic	0.05678	94	1.0335
		Norm	0.05660	94	1.0318
		Student	0.05678	94	nvt
		Bca	0.05660	95	1.0424
		Perc	0.05678	95	1.0335

Tabel 6: Nauwkeurigheid van de PPB en DA bootstrap voor $n = 500$.

We merken op dat dit vrij nauwkeurige betrouwbaarheidsintervallen zijn. De nauwkeurigheid ligt netjes rond de 95% en de gemiddelde lengte is een fractie kleiner dan die van de intervallen gevonden in sectie 6.3. De vraag is echter wat er gebeurt wanneer we de samplegrootte aanpassen.

We kiezen samples van grootte $n = 1000$ en creëren weer 100 95%-betrouwbaarheidsintervallen, dit leidt tot het volgende resultaat:

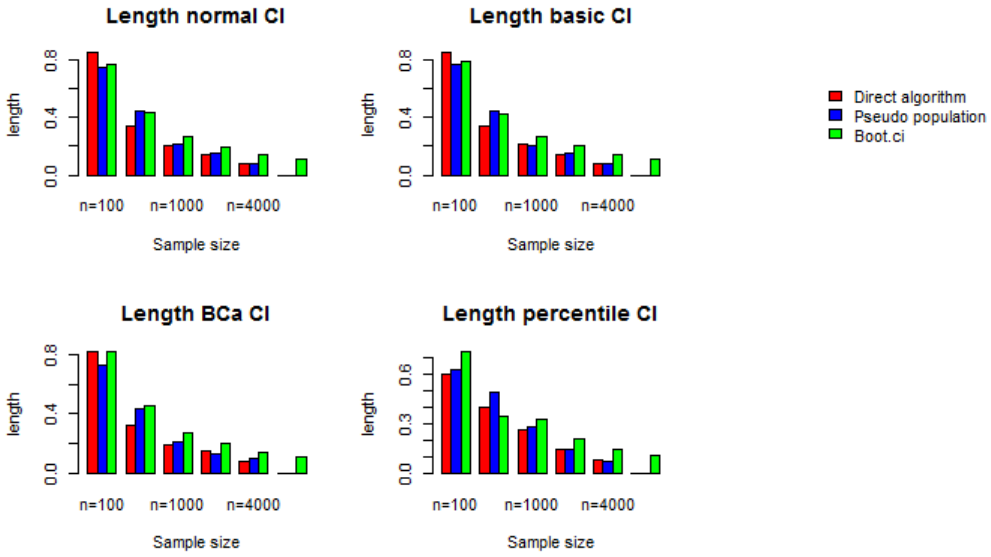
Score	Bootstrap	Type B.I.	Gem. Lengte	Nauwkeurigheid	$\frac{length_{boot.ci}}{length_{finite}}$
MNCS	PPB	Basic	0.2464	96	1.1238
		Norm	0.2455	96	1.1214
		Student	0.2464	96	nvt
		Bca	0.2425	96	1.1674
		Perc	0.2464	97	1.1238
	DA	Basic	0.2485	97	1.1143
		Norm	0.2479	97	1.1105
		Student	0.2485	97	nvt
		Bca	0.2427	97	1.1665
		Perc	0.2484	97	1.1143
$PP_{top10\%}$	PPB	Basic	0.03787	94	1.0897
		Norm	0.03758	93	1.0904
		Student	0.03787	93	nvt
		Bca	0.03751	93	1.1013
		Perc	0.03787	93	1.0897
	DA	Basic	0.03858	96	1.0699
		Norm	0.03836	97	1.0677
		Student	0.03858	96	nvt
		Bca	0.03835	96	1.0755
		Perc	0.03858	97	1.0699

Tabel 7: Nauwkeurigheid van de PPB en DA bootstrap voor $n = 1000$.

We zien dat de nauwkeurigheid nu in de meeste gevallen zelfs boven de 95% ligt. Daarnaast is het verschil in lengte, vergeleken met sectie 6.3 nog groter dan voor $n = 500$.

Om te kijken wat er gebeurt wanneer we onze samples nog groter maken, kiezen we samples van grootte $n \in \{100, 500, 1000, 2000, 4000, N\}$, met $N = 6224$ de populatiegrootte, en creëren voor iedere sample een *normal, basic* en BCa[10] betrouwbaarheidsinterval voor de $MNCS$ met de *boot.ci* functie, de pseudopopulatiemethode en het directe algoritme. Voor iedere methode en voor ieder sample gebruiken we 999 bootstrapreplicaties. Wanneer n te groot wordt voor het BCa interval van *boot.ci*, gebruiken we voor deze methode $2000, 4000$ en 7000 replicaties. Voor elk interval meten we de lengte en vinden de volgende resultaten:

[10]Merkwaardig is dat voor beide methodes $R \geq n$ geen eis is om een BCa BI te creëren, in tegenstelling tot *boot.ci*.

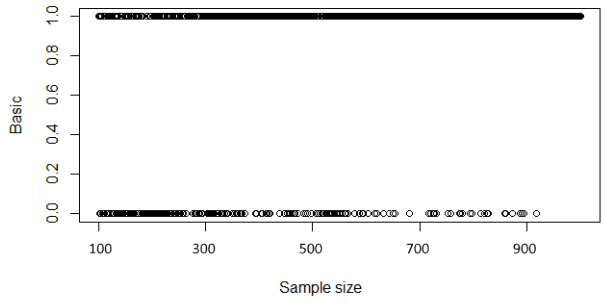


Figuur 8: lengte van verschillende BI bij verschillende samplegrootten

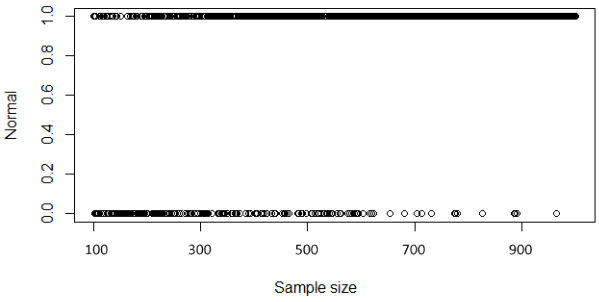
We zien dat de lengtes voor de verschillende methodes voor $n = 100$ en $n = 500$ nog ongeveer gelijk zijn. Echter voor $n = 1000$ zien we de eindige-populatiemethodes onder *boot.ci* terechtkomen, voor $n = N$ verdwijnen de betrouwbaarheidsintervallen zelfs helemaal, zoals we al verwacht hadden: voor $n = N$ is $var(\hat{\theta}^*) = 0$ voor de pseudopopulatiemethode en voor het directe algoritme. De standaardbootstrap denkt nog steeds dat er een kleine fractie van de populatie is gekozen en heeft $var(\hat{\theta}^*) > 0$.

De vraag is nu wanneer de eindige populatie correctie echt verschil gaat maken. Davison en Hinkley[4] (p.92) zeggen dat n/N vaak in het interval $[0.1, 0.5]$ ligt en dan niet genegeerd kan worden. Dit komt overeen met $n \approx 622$ in ons geval en lijkt redelijk te kloppen met bovenstaande figuur. We voeren een test uit door voor verschillende samples van grootte $n \in \{100, 101, \ldots, 1000\}$ een normal, basic en BCa betrouwbaarheidsinterval te creëren met het directe algoritme [11]. We creëren drie vectoren, voor elk interval één. Indien de lengte van het BI van de gebruikte eindige methode kleiner is dan de lengte van het BI van boot.ci, wordt waarde 1 aan de vector toegekend, anders waarde 0, zie 14.6 in de appendix. We vinden de volgende resultaten:

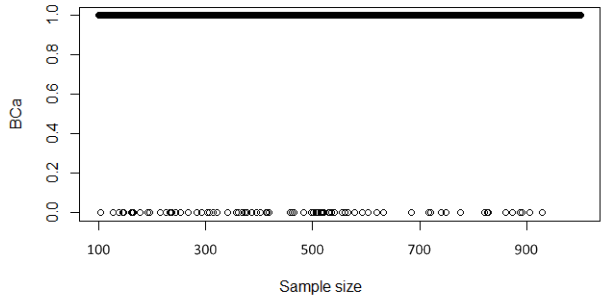
[11] In figuur 8 zien we dat de directe bootstrap en de pseudopopulatie ongeveer hetzelfde reageren. De pseudopopulatie is echter nog erg inefficiënt, vandaar dat we bij deze test de voorkeur aan het directe algoritme geven. De verwachting is dat de PPB hetzelfde resultaat zal geven.



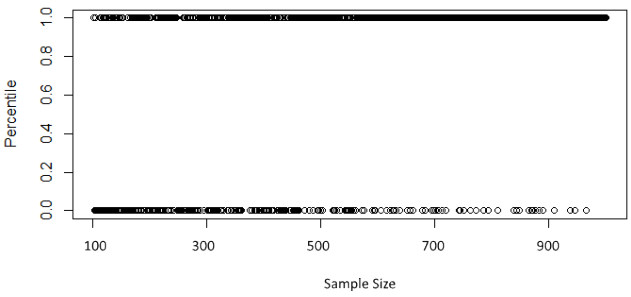
(a) Basic BI



(b) Normal BI



(c) BCa BI



(d) perc BI

Bij de BCa intervallen zien we dat de eindige methode vanaf het begin al kleinere intervallen produceert. Voor de normal, basic en percentile intervallen zien we het aantal keer dat boot.ci een kleiner interval produceert sterk afnemen vanaf $n \approx 600$, wat de woorden van Davison en Hinkley bevestigt: voor $n/N \geq 0.1$ gaat de eindige populatie correctie echt een rol spelen.

13 Conclusie en nader onderzoek

Binnen een populatie waarin de data asymmetrisch verdeeld is, is het belangrijk om over relatief grote samples te beschikken om betrouwbare conclusies over het gemiddelde van de populatie te trekken. Standaard bootstrapmethodes gaan uit van een oneindig grote populatie, waardoor een grote sample door de methode alsnog als klein wordt beschouwd. Een eindige populatie bootstrap zal herkennen dat een relatief grote fractie van de populatie is gekozen, de methode zal hierdoor significant kleinere betrouwbaarheidsintervallen creëren en dus een veel accuratere schatting van een parameter van de populatie kunnen geven.

We hebben gezien dat de standaard bootstrapmethode erg veel moeite heeft om betrouwbaarheidsintervallen van een bepaalde nauwkeurigheid te creëren en dat eindige populatiemethodes met een sampling fraction $f = \frac{n}{N} \geq 0.1$ veel nauwkeuriger intervallen geeft. Binnen de toegepaste statistiek kan dit dus ook veel nauwkeurigere resultaten opleveren.

Bij nader onderzoek zou gekeken kunnen worden naar manieren om de pseudopopulatie bootstrap efficiënter te maken. De implementatie voor dit onderzoek maakt gebruik van erg veel *for-loops* en is daarom erg traag. Ook zou van de implementatie een daadwerkelijke functie in *R* kunnen worden gemaakt, zodat de methodes voor iedereen toegankelijk zijn. Daarnaast is dit onderzoek in zijn geheel gebaseerd op *simple random samples*. Om *multistage* of *stratified sampling* mogelijk te maken, zou onderzoek gedaan moeten worden naar hoe we een extra dimensie aan ons bootstrapalgoritme toe kunnen voegen. Naast *simple random sampling* is ook alleen aandacht besteed aan betrouwbaarheidsintervallen. In de toegepaste statistiek is het testen van hypotheses een erg veel voorkomende onderzoeksmethode. Hier zou het dus ook erg interessant kunnen zijn om rekening te houden met eindige populaties.

14 Appendix

14.1 Boot.ci nauwkeurigheid

```
#Libraries needed
library(MASS)
library(boot)
library(samplingbook)

#Import data, working directory should be different for anyone
setwd("C:\\Users\\kaspe\\OneDrive\\Documenten\\TW\\2015-2016\\Bachelor-eindpro
library(XLConnect)
df2=read.csv("TUDelft_LRdata2.csv",header = TRUE,quote="\"",stringsAsFactors= '

#Select data from period 1.
df2_p1=subset(df2,period_no==1)

#Progress bar
pb=winProgressBar(title = "j-loop progress bar", min = 0, max = 100, width = 3

#accuracy counters
v1=0
v2=0
v4=0
v3=0

#Vectors to store CI-lengths
norm.len=numeric()
basic.len=numeric()
perc.len=numeric()
bca.len=numeric()

for(j in 1:100){
  setWinProgressBar(pb, j, title=paste( round(j/100*100, 0),
                                        "% done"))
  s=df2_p1[sample(1:nrow(df2_p1),500,replace=FALSE),] #sample
  meanFunc <- function(x,i){mean(x[i])}
  boot.out <- boot(s$ncs, meanFunc, R=1000, sim="ordinary")
  all.ci=boot.ci(boot.out,conf=0.95,type="all")

  norm.len[j]=all.ci$normal[3]-all.ci$normal[2]
  basic.len[j]=all.ci$basic[5]-all.ci$basic[4]
  perc.len[j]=all.ci$percent[5]-all.ci$percent[4]
  bca.len[j]=all.ci$bca[5]-all.ci$bca[4]

  if(all.ci$normal[2]<mean(df2_p1$ncs) & mean(df2_p1$ncs)<all.ci$normal[3]){
    v1=v1+1
  }
```

```
  if(all.ci$basic[4]<mean(df2_p1$ncs) & mean(df2_p1$ncs)<all.ci$basic[5]){
    v2=v2+1
  }


  if(all.ci$percent[4]<mean(df2_p1$ncs) & mean(df2_p1$ncs)<all.ci$percent[5]){
    v3=v3+1
  }

  if(all.ci$bca[4]<mean(df2_p1$ncs) & mean(df2_p1$ncs)<all.ci$bca[5]){
    v4=v4+1
  }

}
close(pb)
```

14.2 Boot.ci lengte

```
#Libraries needed
library(MASS)
library(boot)
library(samplingbook)

#Import data, working directory should be different for anyone
setwd("C:\\Users\\kaspe\\OneDrive\\Documenten\\TW\\2015-2016\\Bachelor-eindpro
library(XLConnect)
df2=read.csv("TUDelft_LRdata2.csv",header = TRUE,quote="\"",stringsAsFactors=

#Select data from period 1, calculate the actual mean
df2_p1=subset(df2,period_no==1)
pb <- winProgressBar(title = "j-loop progress bar", min = 0, max = 100, width

v1=0
v2=0
v4=0
v3=0

i=1
norm.len=numeric()
basic.len=numeric()
perc.len=numeric()
bca.len=numeric()

for(j in 100:4000){
  setWinProgressBar(pb, j, title=paste( round(j/1900*100, 0),
                                        "% done"))
  s=df2_p1[sample(1:nrow(df2_p1),j,replace=FALSE),] #sample
  meanFunc <- function(x,i){mean(x[i])}
  boot.out <- boot(s$ncs, meanFunc, R=1000, sim="ordinary")
  all.ci_norm=boot.ci(boot.out,conf=0.95,type="norm")
  all.ci_basic=boot.ci(boot.out,conf=0.95,type="basic")

  norm.len[i]=all.ci_norm$normal[3]-all.ci_norm$normal[2]
  basic.len[i]=all.ci_basic$basic[5]-all.ci_basic$basic[4]

  i=i+1

}
close(pb)


mean(norm.len)
min(norm.len)

mean(basic.len)
```

```
min(basic.len)

plot(norm.len)
abline(h=0.1,col="red",lwd=2)

plot(basic.len)
abline(h=0.1,col="red",lwd=2)
```

14.3 Boot.ci lengte, gehele populatie

```
#Libraries needed
library(MASS)
library(boot)
library(samplingbook)

#Import data, working directory should be different for anyone
setwd("C:\\Users\\kaspe\\OneDrive\\Documenten\\TW\\2015-2016\\Bachelor-eindpro
library(XLConnect)
df2=read.csv("TUDelft_LRdata2.csv",header = TRUE,quote="\"",stringsAsFactors=

#Select data from period 1, calculate the actual mean
df2_p1=subset(df2,period_no==1)
pb <- winProgressBar(title = "j-loop progress bar", min = 0, max = 100, width

v1=0
v2=0
v4=0
v3=0

i=1
norm.len=numeric()
basic.len=numeric()
perc.len=numeric()
bca.len=numeric()

for(j in 0:100){
  setWinProgressBar(pb, j, title=paste( round(j/100*100, 0),
                                        "% done"))
  s=df2_p1[sample(1:nrow(df2_p1),6224,replace=FALSE),] #sample
  meanFunc <- function(x,i){mean(x[i])}
  boot.out <- boot(s$ncs, meanFunc, R=1000, sim="ordinary")
  #all.ci_norm=boot.ci(boot.out,conf=0.95,type="norm")
  #all.ci_basic=boot.ci(boot.out,conf=0.95,type="basic")
   all.ci_perc=boot.ci(boot.out,conf=0.95,type="perc")

  norm.len[j]=all.ci_norm$normal[3]-all.ci_norm$normal[2]
  basic.len[j]=all.ci_basic$basic[5]-all.ci_basic$basic[4]
  perc.len[j]=all.ci_perc$percent[5]-all.ci_perc$percent[4]

}
close(pb)


mean(norm.len)
min(norm.len)

mean(basic.len)
```

```
min(basic.len)

mean(perc.len)
min(perc.len)

plot(norm.len)
abline(h=0.102,col="red",lwd=2)

plot(basic.len)
abline(h=0.1,col="red",lwd=2)

plot(perc.len)
abline(h=0.1,col="red",lwd=2)
```

14.4 Pseudopopulatiebootstrap

```
#Simple random sampling without replacement, pseudo population bootstrap metho

#Libraries needed
library(MASS)
library(boot)
library(samplingbook)

#import data, working directory should be different for anyone
setwd("C:\\Users\\kaspe\\OneDrive\\Documenten\\TW\\2015-2016\\Bachelor-eindpro
library(XLConnect)
df2=read.csv("TUDelft_LRdata2.csv",header = TRUE,quote="\"",stringsAsFactors= "

norm.len=numeric()
ba.len=numeric()
bca.len=numeric()
st.len=numeric()
perc.len=numeric()

#Select data from period 1
df2_p1=subset(df2,period_no==1)

N=length(df2_p1$ncs)       #Population size
n=500

#Sample size
B=999                      #Amount of bootstrap replications
D=999                      #Amount of pseudo populations
k=floor(N/n)               #Every unit in the original sample is repeated k tim
m=mean(df2_p1$ncs)         #The actual mean from ncs

pb <- winProgressBar(title = "z-loop progress bar", min = 0, max = 100, width =
pb2 <- winProgressBar(title = "j-loop progress bar", min = 0, max = 100, width

v8=0
v9=0
v10=0
v11=0
v12=0

for(z in 1:100){
  setWinProgressBar(pb, z, title=paste( round(z/100*100, 0),
                                        "% done"))


  s=df2_p1[sample(1:nrow(df2_p1),n,replace=FALSE),] #sample
  s.mean=mean(s$ncs)
```

```
#Create vectors to store bootstrap estimates.
PPboot.mean=numeric(B)
mean.boot=numeric(B)
boot.var=numeric(D)

#Step 1: repeat each unit in original sample k times to create the fixed par
U.f=rep(s$ncs,k)

#Step 6: repeat steps 2 to 5 D times
for(j in 1:D){
  setWinProgressBar(pb2, j, title=paste( round(j/D*100, 0),
                                         "% done"))

  #Step 2: draw U.cboot from s to complete the pseudo-population U.boot
  U.cboot=sample(s$ncs,N-n*k,replace=FALSE)
  U.boot=c(U.f,U.cboot)
  PPm.boot=mean(U.boot)
  PPboot.mean[j]=PPm.boot

  #Step 5': Repeat steps 3 and 4 B times.
  for(i in 1:B){
    #Step 3: take a SRS, s.boot, of size n w.o. replacement from U.boot
    s.boot=sample(U.boot,n,replace=FALSE)
    #Step 4: create the bootstrap statistic m_est.boot on the bootstrap samp
    m_est.boot=mean(s.boot)
    mean.boot[i]=m_est.boot
  }

  #Calculate the bootstrapped mean and variance, see (4.2) or step 5' of Z.M
  m.boot.=1/B*sum(mean.boot)
  Bootvar.est1=sum((mean.boot-m.boot.)^2)/(B-1)
  boot.var[j]=Bootvar.est1
}

#The final estimate of our bootstrap variance, see step 6 of Z. Mashregi et
bootvar.est.fin=sum(boot.var)/D
mean.boot.sort=sort(mean.boot)

  #Basic confidence interval
  a.ba_25=2*mean(s$ncs)-mean.boot.sort[(B+1)*(1-0.025)]
  a.ba_975=2*mean(s$ncs)-mean.boot.sort[(B+1)*(0.025)]
  ci_ba=c(a.ba_25,a.ba_975)

  #Normal confidence interval
  b_R=mean(mean.boot)-s.mean
  a.norm_25=s.mean-b_R-sqrt(bootvar.est.fin)*qnorm(1-0.025)
  a.norm_975=s.mean-b_R+sqrt(bootvar.est.fin)*qnorm(1-0.025)
  ci_norm=c(a.norm_25,a.norm_975)
```

```
    #Studentized
    z.boot=(mean.boot-s.mean)/sqrt(bootvar.est.fin)
    z.boot.sort=sort(z.boot)
    a.st_25=s.mean-b_R-sqrt(bootvar.est.fin)*z.boot.sort[(B+1)*(1-0.025)]
    a.st_975=s.mean-b_R-sqrt(bootvar.est.fin)*z.boot.sort[(B+1)*(0.025)]
    ci_st=c(a.st_25,a.st_975)

    #BCa
    l1=mean.boot-mean(mean.boot)
    a=(1/6)*(sum(l1^3))/((sum(l1^2))^(3/2))
    w=qnorm(sum(mean.boot<=s.mean)/(B+1))
    a_est=pnorm(w+(w+qnorm(0.025))/(1-a*(w+qnorm(0.025))))
    a.bca_25=mean.boot.sort[(B+1)*a_est]
    a.bca_975=mean.boot.sort[(B+1)*(1-a_est)]
    ci_bca=c(a.bca_25,a.bca_975)

    #95% percentile conf. int
    a.perc_25=mean.boot.sort[(B+1)*0.025]
    a.perc_975=mean.boot.sort[(B+1)*(1-0.025)]
    ci_perc=c(a.perc_25,a.perc_975)

    #lengths of confidence intervals
    norm.len[z]=a.norm_975-a.norm_25
    ba.len[z]=a.ba_975-a.ba_25
    bca.len[z]=a.bca_975-a.bca_25
    st.len[z]=a.st_975-a.st_25
    perc.len[z]=a.perc_975-a.perc_25

    if(a.ba_25<m & m<a.ba_975){
      v8=v8+1
    }

    if(a.norm_25<m & m<a.norm_975){
      v9=v9+1
    }
    if(a.st_25<m & m<a.st_975){
      v10=v10+1
    }

    if(a.bca_25<m & m<a.bca_975){
      v11=v11+1
    }

    if(a.perc_25<m & m <a.perc_975){
      v12=v12+1
    }
}
```

14.5 Direct Algoritme

```
#Simple random sampling without replacement, direct algorithm, by Sitter (1992

#Libraries needed
library(MASS)
library(boot)
library(samplingbook)

#Import data, working directory should be different for anyone
setwd("C:\\Users\\kaspe\\OneDrive\\Documenten\\TW\\2015-2016\\Bachelor-eindpro
library(XLConnect)
df2=read.csv("TUDelft_LRdata2.csv",header = TRUE,quote="\"",stringsAsFactors= '

#Select data from period 1, calculate the actual mean
df2_p1=subset(df2,period_no==1)
m=mean(df2_p1$p_top_10)
pb <- winProgressBar(title = "j-loop progress bar", min = 0, max = 100, width =
N=length(df2_p1$p_top_10) #Population size

v8=0
v9=0
v10=0
v11=0
v12=0



n=1000                    #Mashreghi et al uses both n and n1 for the sample
n1=1000
f=n/N                     #sampling fraction
C=1                       #Rescaling factor of the observations
p=900
n2=(n-p)/(2-f)            #n2 should be <n/(2-f). Simulation shows that our c.
                          #Probably for all p>400, computation time increases
B=999                     #Amount of bootstrap replicates.

#Arguments for Sitter (1992b)
f2=n2/n
k=((n*(1-f2))/(n2*(1-f)))
q=(floor(k)^(-1)-k^(-1))/(floor(k)^(-1)-ceiling(k)^(-1))
k1=floor((n*(1-f2))/(n2*(1-f)))+rbinom(1,1,q)

norm.len=numeric()
ba.len=numeric()
bca.len=numeric()
st.len=numeric()
perc.len=numeric()
```

```
for(z in 1:100){
  setWinProgressBar(pb, z, title=paste( round(z/100*100, 0),
                                        "% done"))

  s=df2_p1[sample(1:nrow(df2_p1),n1,replace=FALSE),] #sample
  s.mean=mean(s$p_top_10) #sample mean

  s1=numeric(n)           #vector to create bootstrap samples
  mean.boot=numeric(B)    #Vector to store bootstrap estimates

  #Step 1 SRSWOR Direct Algorithm
  for(i in 1:n){
    s1[i]=s.mean+C*(s$p_top_10[i]-s.mean)
  }

  #Step 5: repeat steps 2 to 4 B times.
  for(l in 1:B){
    s.boot=numeric()

    #Step 3: repeat step 2 k1 times independently.
    for(j in 1:k1){
      #Step 2: take a simple random sample of size n2 without replacement fro
      srs=sample(s1,n2,replace=FALSE)
      #Step 3.1: concatenate all subsamples to get s.boot
      s.boot=c(s.boot,srs)
    }
    #step 4: compute the bootstrap statistic. Mashregi et al uses a horvitz-
    #This shouldn't be necessary when calculating the mean
    mean.boot[l]=mean(s.boot)
  }

  #Step 6: estimate the variance of our bootstrap statistic
  m.boot.est=sum(mean.boot)/B
  V.boot.est=sum((mean.boot-m.boot.est)^2)/(B-1)

  #percentile C.I.
  boot.mean.sort=sort(mean.boot)

  #Basic confidence interval
  a.ba_25=2*mean(s$p_top_10)-boot.mean.sort[(B+1)*(1-0.025)]
  a.ba_975=2*mean(s$p_top_10)-boot.mean.sort[(B+1)*(0.025)]
  ci_ba=c(a.ba_25,a.ba_975)

  #Normal confidence interval
  b_R=mean(mean.boot)-s.mean
  a.norm_25=s.mean-b_R-sqrt(V.boot.est)*qnorm(1-0.025)
  a.norm_975=s.mean-b_R+sqrt(V.boot.est)*qnorm(1-0.025)
  ci_norm=c(a.norm_25,a.norm_975)
```

```
  #Studentized
  z.boot=(mean.boot-s.mean)/sqrt(V.boot.est)
  z.boot.sort=sort(z.boot)
  a.st_25=s.mean-b_R-sqrt(V.boot.est)*z.boot.sort[(B+1)*(1-0.025)]
  a.st_975=s.mean-b_R-sqrt(V.boot.est)*z.boot.sort[(B+1)*(0.025)]
  ci_st=c(a.st_25,a.st_975)

  #BCa
  l1=mean.boot-mean(mean.boot)
  a=(1/6)*(sum(l1^3))/((sum(l1^2))^(3/2))
  w=qnorm(sum(mean.boot<=s.mean)/(B+1))
  a_est=pnorm(w+(w+qnorm(0.025))/(1-a*(w+qnorm(0.025))))
  a.bca_25=boot.mean.sort[(B+1)*a_est]
  a.bca_975=boot.mean.sort[(B+1)*(1-a_est)]
  ci_bca=c(a.bca_25,a.bca_975)

  #percentile C.I.
  a.perc_25=boot.mean.sort[(B+1)*0.025]
  a.perc_975=boot.mean.sort[(B+1)*(1-0.025)]
  ci_perc=c(a.perc_25,a.perc_975)


  #lengths of confidence intervals
  norm.len[z]=a.norm_975-a.norm_25
  ba.len[z]=a.ba_975-a.ba_25
  bca.len[z]=a.bca_975-a.bca_25
  st.len[z]=a.st_975-a.st_25
  perc.len[z]=a.perc_975-a.perc_25

  if(a.ba_25<m & m<a.ba_975){
    v8=v8+1
  }

  if(a.norm_25<m & m<a.norm_975){
    v9=v9+1
  }
  if(a.st_25<m & m<a.st_975){
    v10=v10+1
  }

  if(a.bca_25<m & m<a.bca_975){
    v11=v11+1
  }

  if(a.perc_25<m & m < a.perc_975){
    v12=v12+1
  }
}
```

14.6 Significantie sampling fraction, direct algoritme

```
#Simple random sampling without replacement, direct algorithm, by Sitter (1992

#Libraries needed
library(MASS)
library(boot)
library(samplingbook)

#Import data, working directory should be different for anyone
setwd("C:\\Users\\kaspe\\OneDrive\\Documenten\\TW\\2015-2016\\Bachelor-eindpro
library(XLConnect)
df2=read.csv("TUDelft_LRdata2.csv",header = TRUE,quote="\"",stringsAsFactors=

#Select data from period 1, calculate the actual mean
df2_p1=subset(df2,period_no==1)
m=mean(df2_p1$ncs)
pb <- winProgressBar(title = "j-loop progress bar", min = 0, max = 100, width
N=length(df2_p1$ncs) #Population size

v8=numeric()
v9=numeric()
v10=numeric()
j=1

for(z in 100:1000){

n=z                        #Mashreghi et al uses both n and n1 for the sample siz
n1=z
f=n/N                        #sampling fraction
C=1                          #Rescaling factor of the observations
p=z-0.2*z
n2=(n-p)/(2-f)               #n2 should be <n/(2-f). Simulation shows that our c.
                             #Probably for all p>400, computation time increases
B=999                        #Amount of bootstrap replicates.

#Arguments for Sitter (1992b)
f2=n2/n
k=((n*(1-f2))/(n2*(1-f)))
q=(floor(k)^(-1)-k^(-1))/(floor(k)^(-1)-ceiling(k)^(-1))
k1=floor((n*(1-f2))/(n2*(1-f)))+rbinom(1,1,q)


    setWinProgressBar(pb, z, title=paste( round(z/900*100, 0),
                                          "% done"))

    s=df2_p1[sample(1:nrow(df2_p1),n1,replace=FALSE),] #sample
    s.mean=mean(s$ncs) #sample mean
```

```
s1=numeric(n)            #vector to create bootstrap samples
mean.boot=numeric(B)     #Vector to store bootstrap estimates

#Step 1 SRSWOR Direct Algorithm
for(i in 1:n){
  s1[i]=s.mean+C*(s$ncs[i]-s.mean)
}

#Step 5: repeat steps 2 to 4 B times.
for(l in 1:B){
  s.boot=numeric()

  #Step 3: repeat step 2 k1 times independently.
  for(z in 1:k1){
    #Step 2: take a simple random sample of size n2 without replacement fro
    srs=sample(s1,n2,replace=FALSE)
    #Step 3.1: concatenate all subsamples to get s.boot
    s.boot=c(s.boot,srs)
  }
  #step 4: compute the bootstrap statistic. Mashregi et al uses a horvitz-
  #This shouldn't be necessary when calculating the mean
  mean.boot[l]=mean(s.boot)
}

#Step 6: estimate the variance of our bootstrap statistic
m.boot.est=sum(mean.boot)/B
V.boot.est=sum((mean.boot-m.boot.est)^2)/(B-1)

#percentile C.I.
boot.mean.sort=sort(mean.boot)

#Boot.ci function
meanFunc <- function(x,i){mean(x[i])}
boot.out <- boot(s$ncs, meanFunc, R=1000, sim="ordinary")
all.ci=boot.ci(boot.out,conf=0.95,type="all")

norm.len.ci=all.ci$normal[3]-all.ci$normal[2]
ba.len.ci=all.ci$basic[5]-all.ci$basic[4]
perc.len.ci=all.ci$percent[5]-all.ci$percent[4]
bca.len.ci=all.ci$bca[5]-all.ci$bca[4]

#Basic confidence interval
a.ba_25=2*mean(s$ncs)-boot.mean.sort[(B+1)*(1-0.025)]
a.ba_975=2*mean(s$ncs)-boot.mean.sort[(B+1)*(0.025)]
ci_ba=c(a.ba_25,a.ba_975)

#Normal confidence interval
b_R=mean(mean.boot)-s.mean
a.norm_25=s.mean-b_R-sqrt(V.boot.est)*qnorm(1-0.025)
```

```
  a.norm_975=s.mean-b_R+sqrt(V.boot.est)*qnorm(1-0.025)
  ci_norm=c(a.norm_25,a.norm_975)

  #Studentized
  z.boot=(mean.boot-s.mean)/sqrt(V.boot.est)
  z.boot.sort=sort(z.boot)
  a.st_25=s.mean-b_R-sqrt(V.boot.est)*z.boot.sort[(B+1)*(1-0.025)]
  a.st_975=s.mean-b_R-sqrt(V.boot.est)*z.boot.sort[(B+1)*(0.025)]
  ci_st=c(a.st_25,a.st_975)

  #BCa
  l1=mean.boot-mean(mean.boot)
  a=(1/6)*(sum(l1^3))/((sum(l1^2))^(3/2))
  w=qnorm(sum(mean.boot<=s.mean)/(B+1))
  a_est=pnorm(w+(w+qnorm(0.025))/(1-a*(w+qnorm(0.025))))
  a.bca_25=boot.mean.sort[(B+1)*a_est]
  a.bca_975=boot.mean.sort[(B+1)*(1-a_est)]
  ci_bca=c(a.bca_25,a.bca_975)

  #lengths of confidence intervals
  norm.len=a.norm_975-a.norm_25
  ba.len=a.ba_975-a.ba_25
  bca.len=a.bca_975-a.bca_25
  st.len=a.st_975-a.st_25

  if(norm.len<norm.len.ci){
    v8[j]=1
  }
  else{ v8[j]=0}

  if(ba.len<ba.len.ci){
    v9[j]=1
  }
  else{ v9[j]=0}

  if(bca.len<bca.len.ci){
    v10[j]=1
  }
  else{ v10[j]=0}

  j=j+1
}
```

Referenties

[1] Booth, J. G., Ronald W. Butler, Peter Hall (1994). *Bootstrap Methods for Finite Populations.* Journal of the American Statistical Association, Vol. 89, No. 428. (Dec., 1994), pp. 1282-1289

[2] Carpenter, J, and John Bithell *Bootstrap confidence intervals: when, which, what? A practical guide for medical statisticians.* Medical Statistics Unit, London School of Hygiene and Tropical Medicine, 1999

[3] Chauvet, G. (2007). *Méthodes de bootstrap en population finie.* Ph. D. thesis, Université de Rennes 2.

[4] Davison, A.C., and Hinkley, D.V. *Bootstrap Methods and their Application* Cambridge Series in Statistical and Probabilistic Mathematics

[5] Dekking, F.M., C Kraaikamp, H.P. Lopuhaä, L.E. Meester *A Modern Introduction to Probability and Statistics*, page 202

[6] Efron, B. *Bootstrap methods: another look at the jackknife.* The Annals of Statistics 7(1), 1-26

[7] Hajek, J *Sampling from a finite population* Statistics: textbooks and monographs, volume 37

[8] Ludo Waltman, Clara Calero-Medina, Joost Kosten, Ed C.M. Noyons, Robert J.W. Tijssen, Nees Jan van Eck, Thed N. van Leeuwen, Anthony F.J. van Raan, Martijn S. Visser, and Paul Wouters *The Leiden Ranking 2011/2012: Data Collection, Indicators, and Interpretation.* Journal of the American society for information science and technology, 2012

[9] Mashreghi, Z., David Haziza, and Christian Léger (2016). *A survey of bootstrap methods in finite population sampling.* Statistic Surveys Vol. 10, 1-52

[10] Sitter, R.R. (1992). *A Resampling Procedure for Complex Survey Data.* Journal of the American Statistical Association, 87:419, 755-765

[11] *The Bootstrap* 36-402, Advanced Data Analysis, 3 February 2011