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

Het schatten van parameters van een marktmodel met Approximate Bayesian Computational methoden.

(Engelse titel: Parameter estimation of a trading model by Approximate Bayesian Computational methods.)

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

JOSEPHINE ALBERTS

Delft, Nederland
Juli 2013

BSc verslag TECHNISCHE WISKUNDE

"Het schatten van parameters van een marktmodel met Approximate Bayesian Computional methoden

(Engelse titel: "Parameter estimation of a trading model by Approximate Bayesian Computational methods.)"

JOSEPHINE ALBERTS

Technische Universiteit Delft

Begeleider

Dr. F.H. van der Meulen

Overige commissieleden

Dr. M.B. van Gijzen

Dr. B. van den Dries

Juli, 2013

Delft

Inhoudsopgave

Hoofdstuk 1

Inleiding

Om te onderzoeken hoe je aandelenkoersen kunt beschrijven aan de hand van wiskundige formules bekijken we een model dat Ghoulmie, Cont en Nadal [1] in 2005 hiervoor opgesteld hebben. Dit eenvoudige model beschrijft de prijs van een aandeel in een markt met daarin alleen dat aandeel, deze wordt verhandeld door N handelaren.

Vervolgens vergelijken we de resultaten van dit model met twee echte aandelenkoersen aan de hand van het boek Financial risk forecasting [2]. Dit om meer te weten te komen over veel voorkomende eigenschappen van aandelenkoersen en om te kijken of we deze eigenschappen ook terug zien in de resultaten van het model.

Stel dan dat we een bepaalde aandelenkoers hebben en we willen weten welke waarden we de parameters van het model moeten geven om het model zo goed mogelijk aan te passen aan deze koers. We gaan proberen parameters van het model te schatten aan de hand van een dataset. De meeste bekende methoden om parameters van een model te schatten maken gebruik van de likelihoodfunctie, de kans op de dataset als functie van de parameter. Helaas is het voor ons model, net als voor veel andere modellen ook in andere vakgebieden, niet mogelijk om een likelihoodfunctie op te stellen. Om toch de parameters te kunnen schatten gebruiken we een relatief nieuwe methode, een Approximate Bayesian Computation (ABC) methode. Deze gebruikt geen likelihoodfunctie bij het schatten van parameters.

Vervolgens bekijken we twee variaties op de standaard ABC methode. De eerste construeert een Markovketen die onder de juiste omstandigheden convergeert naar de verdeling die de parameter zou hebben op grond van de dataset. Omdat de rekentijd bij zowel de standaard ABC methode als bij de eerste variatie hierop lang is, bekijken we als laatste een tweede variatie die sneller een goed resultaat kan opleveren.

Hoofdstuk 2

Model

2.1 Beschrijving van het model

We beschouwen een model van Ghoulmie, Cont en Nadal [1]. Dit model beschrijft een financiële markt met een enkel aandeel met prijs p_t op tijdstip t dat wordt verhandeld door N handelaren. Het handelen vindt plaats op de discrete tijdstippen $t = 0, 1, 2, ..., T$, die gezien kunnen worden als dagen. De vraag van handelaar i ($i = 0, 1, 2, ..., N$) op tijdstip t wordt aangegeven door $\phi_i(t)$. Op elk tijdstip kunnen de handelaren een aandeel kopen ($\phi_i(t) = 1$), een aandeel verkopen ($\phi_i(t) = -1$) of niets doen ($\phi_i(t) = 0$). De informatiestroom ϵ_t op basis waarvan handelaren een beslissing maken wordt gemodelleerd door een rij van onafhankelijke stochastische variabelen elk met verdeling $N(0, D^2)$. De ϵ_t is een voorspelling voor het toekomstige rendement r_t en wordt door alle handelaren ontvangen. Om hun beslissing te maken vergelijken ze de ϵ_t met hun eigen handelsregel, gegeven door een bepaalde drempelwaarde die veranderd in de tijd, $\theta_i(t)$. Als $|\epsilon_t| \leq \theta_i(t)$ zal de handelaar i niets doen, als $\epsilon_t > \theta_i(t)$ zal de handelaar een aandeel kopen en als $\epsilon_t < -\theta_i(t)$ zal de handelaar een aandeel vekopen. De vraag van handelaar i wordt daarom gegeven door:

$$\phi_i(t) = 1_{\epsilon_t > \theta_i(t)} - 1_{\epsilon_t < -\theta_i(t)} \tag{2.1}$$

De variabele Z_t die het oveschot of tekort aan vraag aangeeft wordt gegeven door $Z_t = \sum_{i=1}^{N} \phi_i(t)$. Als deze niet nul is heeft dit een verandering in de prijs tot gevolg gegeven door

$$r_t = \ln\left(\frac{p_t}{p_{t-1}}\right) = g\left(\frac{Z_t}{N}\right) \tag{2.2}$$

met $g : \mathbf{R} \mapsto \mathbf{R}$ de stijgende prijs impact functie met $g(0) = 0$. De marktdiepte λ wordt gedefinieerd door $g'(0) = \frac{1}{\lambda}$.
Eén van de parameters is de beginverdeling F_0 voor de drempelwaarden, $\theta_i(0)$, $i = 1, ..., N$ zijn positieve IID variabelen getrokken uit F_0. Het updaten van de drempelwaarden gebeurt niet allemaal tegelijkertijd, op elk tijdstip t heeft handelaar i een kans van $0 \leq s \leq 1$ dat hij zijn drempelwaarde verandert. Dus in een grote populatie handelaren is s de fractie van de handelaren dat zijn drempelwaarde op tijdstip t update. Hieruit volgt dat $1/s$ is de gemiddelde periode dat een handelaar een bepaalde drempelwaarde $\theta_i(t)$ aanhoudt.

Als een handelaar zijn drempelwaarde verandert, wordt de nieuwe drempelwaarde de geobserveerde absolute waarde van het rendement op tijdstip $t-1$, $|r_{t-1}| = |\ln(\frac{p_{t-1}}{p_{t-2}})|$, dit is een indicator voor de recente volatiliteit. Als $u_i(t)$ onafhankelijk verdeelde stochastische variabelen zijn met elk een uniforme verdeling op $[0,1]$, die aanduiden of handelaar i zijn drempelwaarde update, dan worden de drempelwaarden gegeven door:

$$\theta_i(t) = 1_{u_i(t)<s} * |r_{t-1}| + 1_{u_i(t)\geq s} * \theta_i(t-1) \tag{2.3}$$

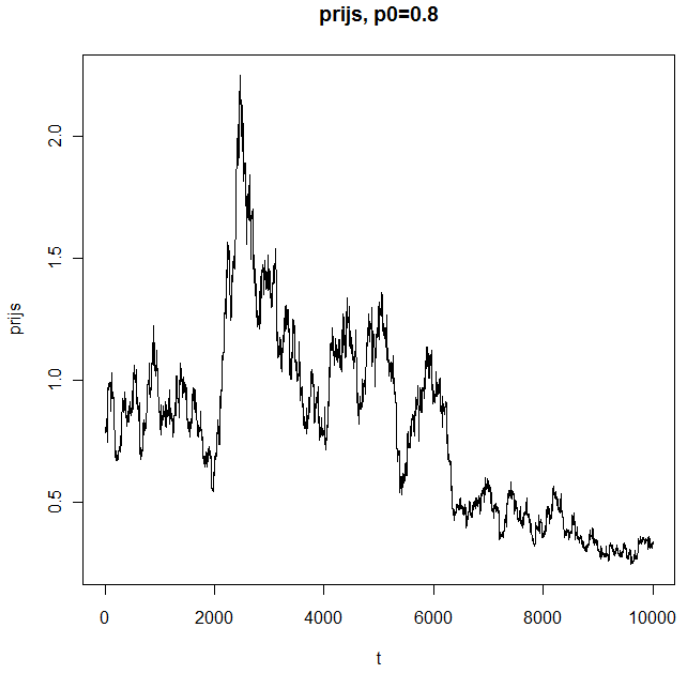
2.2 Simulatie van het model in R

Net als deze simulatie van het model zijn alle implementaties gedaan in R. Dit softwarepakket wordt veel gebruikt bij data-analyse en statistische toepassingen. De simulatie van het model gaat als volgt:

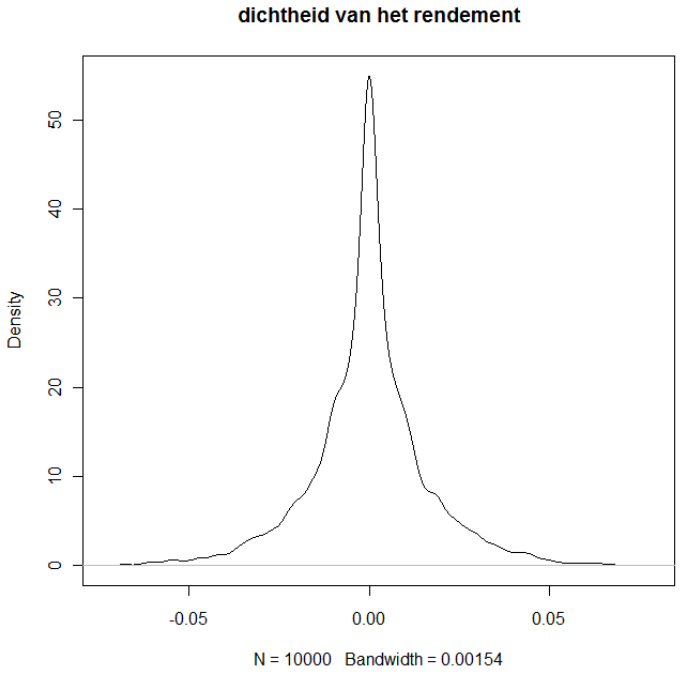
1. Kies om echte datasets na te bootsen de waarden $T = 10000, N = 1500$, $s = 0.1, D = 0.001, \lambda = 10, g(z) = \frac{z}{\lambda}$, en $\theta_i(0) = \theta_0 = \frac{1}{2}$, [1].
2. Simuleer $t * N$ keer een $U(0,1)$ verdeling en vul hiermee de matrix $u_i(t)$.
3. Simuleer t keer een $N(0, D^2)$ verdeling en vul de vector ϵ_t.
4. Vul de eerste rij van de matrix θ op met de beginverdeling voor de $\theta_i(0)$.
5. Bereken de eerste tijdstap van ϕ en de eerste waarde van Z en van r.
6. Bereken vervolgens met behulp van een dubbele for-loop eerst voor elke rij (en dus voor elke tijdstap) de θ voor elke handelaar (elke kolom staat voor een handelaar) en hiermee de ϕ voor deze handelaar.
7. Gebruik deze $\phi's$ om Z en r te berekenen voor elke tijdstap.
8. Bereken en plot met behulp van (2.2) de prijs $p_t = \exp r_t * p_{t-1}$ van een aandeel met beginprijs p_0.
9. Plot de dichtheid van het rendement en de marktactiviteit (aantal handelaren dat per tijdstip koopt of verkoopt).

2.3 Resultaten van de simulatie

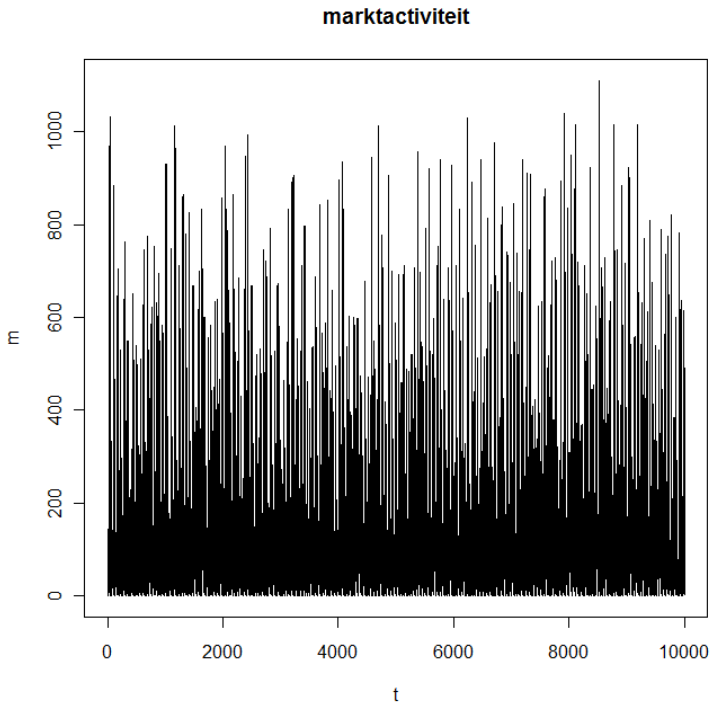
Zie figuur-2.1, figuur-2.2, figuur-2.3 en figuur-2.4 voor de resultaten van de simulatie.



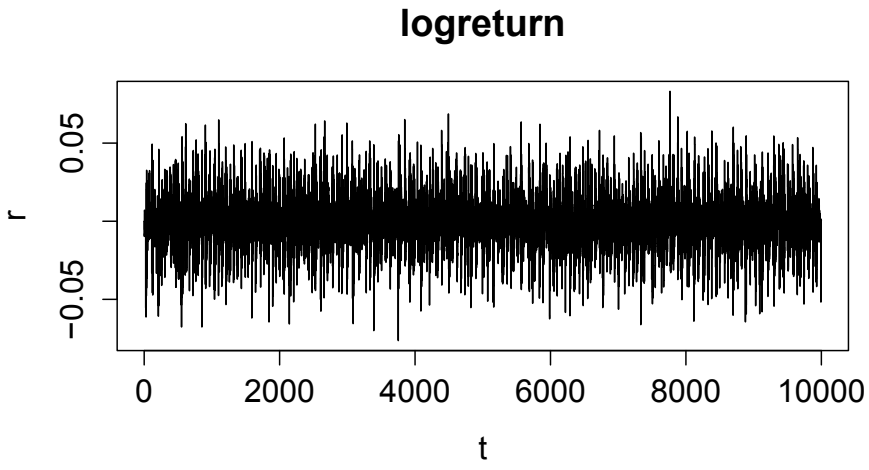
Figuur 2.1: $t \mapsto p_t, p_o = 0.8$



Figuur 2.2: kansdichtheids-schatter op grond van $r_1, ..., r_t$



Figuur 2.3: marktactiviteit m, het aantal handelaren dat per tijdstap koopt of verkoopt



Figuur 2.4: $t \mapsto r_t$

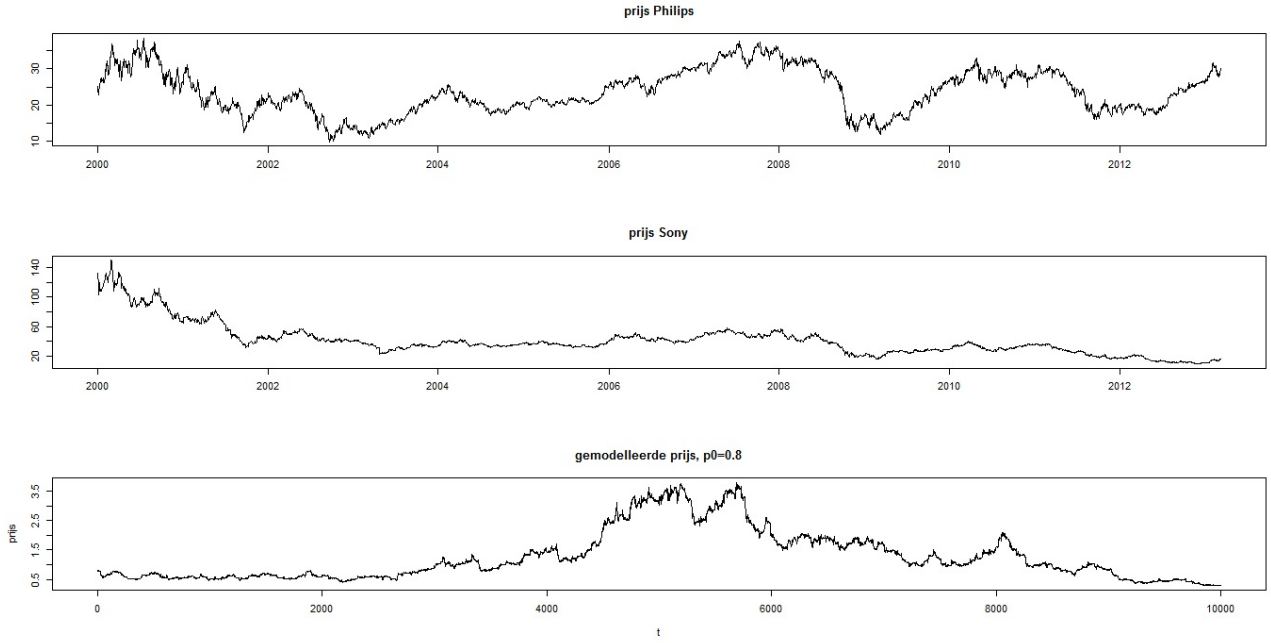
Hoofdstuk 3

Analyse van de simulaties

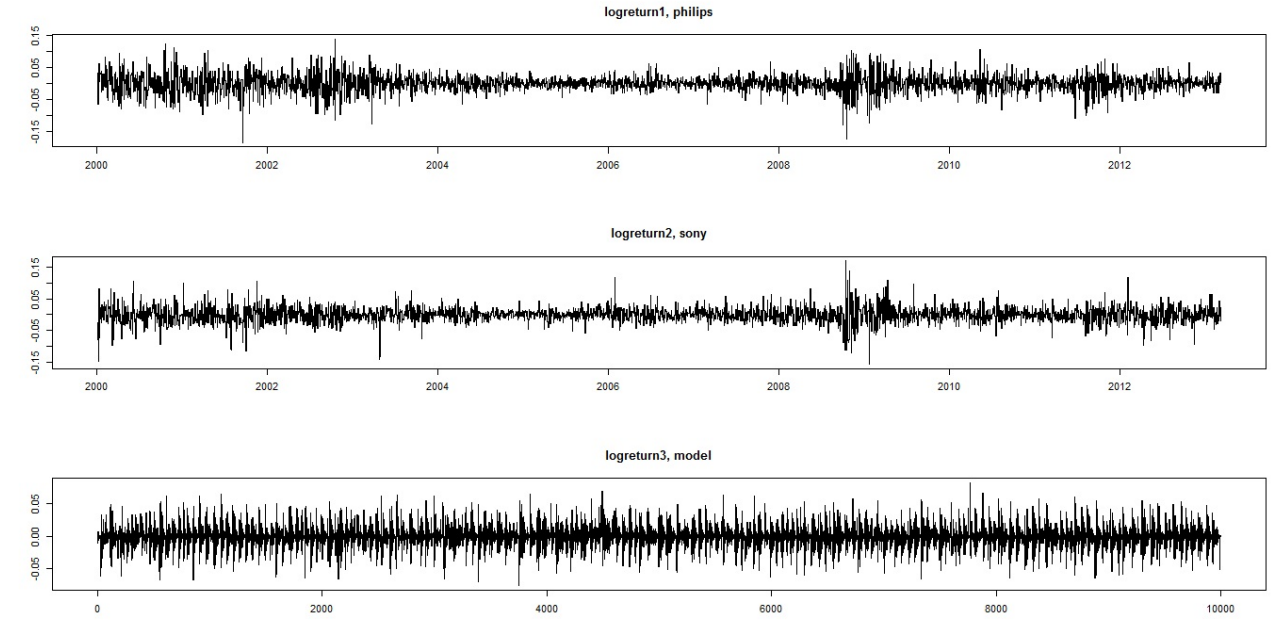
Om de resultaten van het model van Ghoulmie, Cont en Nadal te onderzoeken worden de logreturns, gegenereerd met het model, met echte data vergeleken. Dit aan de hand van methoden uit het boek Financial risk forecasting [2]. Als vergelijkingsdata zijn genomen de aangepaste slotprijs van het aandeel Philips en van het aandeel Sony in de periode januari 2000 tot maart 2013. Deze prijs is de slotprijs die aangepast is voor acties die de waarde van het aandeel beinvloeden die gebeuren voor de volgende handelsdag. Deze prijs wordt over het algemeen gebruikt om historische financiële data te analyseren. Van deze prijs wordt voor zowel Philips als voor Sony het rendement berekend, oftewel de logreturn op tijdstip t:

$$r_t = \ln\left(\frac{p_t}{p_{t-1}}\right)$$

De prijzen van de aandelen Philips en Sony en een product met beginprijs 0.8 euro in het model en de logreturns van deze drie worden gegeven in figuur-3.1 en figuur-3.2. Het is visueel direct duidelijk dat de logreturns die gesimuleerd zijn met het model een bepaalde regelmatigheid vertonen die niet terug te zien is in de logreturns van de echte aandelenkoersen.



Figuur 3.1: prijzen van de aandelen en een gemodelleerd product



Figuur 3.2: logreturns van de aandelen en een gemodelleerd product

3.1 Statistische eigenschappen

Nadat we de prijzen van de aandelen hebben omgezet in logreturns, kunnen we hiervan een aantal statistische eigenschappen bekijken. De logreturns geven de relatieve verandering van de prijs van een aandeel en kunnen beter met elkaar vergeleken worden dan de prijzen zelf. Zie tabel-3.1 voor een aantal statistische eigenschappen die berekend zijn voor de verschillende logreturns.

	Philips	**Sony**	**model**
gemiddelde	0,0000370	-0,000647	-0,000105
standaarddeviatie	0,0273	0,0241	0,0164
minimum	-0,185	-0,155	-0,0764
maximum	0,140	0,170	0,0831
scheefheid	-0,201	-0,0756	-0,0180
kurtosis	6,27	7,74	4,82
autocorrelatie (tijdsvertraging 1) logreturns	-0,042	0,001	-0,001
autocorrelatie (tijdsvertraging 1) gekwadrateerde logreturns	0,205	0,226	0,230
Jarque-Bera (p-waarde)	0,00	0,00	0,00
Ljung-Box t/m tijdsvertraging 20, logreturns (p-waarde)	0,00	0,0720	0,0348
Ljung-Box t/m tijdsvertraging 20, gekwadrateerde logreturns (p-waarde)	0,00	0,00	0,00

Tabel 3.1: statistische eigenschappen logreturns

Uit de waarden in de tabel zien we dat voor alledrie de logreturns het gemiddelde heel klein is, zeker in vergelijking met de standaarddeviatie. Dit zorgt ervoor dat we kunnen aannemen dat deze nul is zonder dat dit grote problemen oplevert. De standaarddeviatie gebruiken we als maat voor de volatiliteit van de logreturns, dus deze zegt iets over de bewegelijkheid van de koers van het aandeel.

De kurtosis en scheefheid van een stochastische variabele X worden respectievelijk gegeven door, [3]

$$K = \frac{E[(X-\mu)^4]}{(E[(X-\mu)^2])^2},\ S = \frac{E[(X-\mu)^3]}{(E[(X-\mu)^2])^{\frac{3}{2}}}$$

. Op grond van een dataset $x_1, ..., x_n$ kunnen deze geschat worden met

$$K_n = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^4}{(\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^2)^2},\ S_n = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^3}{(\frac{1}{n}\sum_{i=1}^{n}(x_i-\overline{x})^2)^{\frac{3}{2}}}$$

De kurtosis is een maat voor piekvormigheid of 'platheid' in een verdeling en de scheefheid is een maat voor asymmetrie in een verdeling. Uit de waarden voor minimum, maximum en kurtosis kunnen we zien dat ons model over het algemeen minder extreme returns geeft dan de prijzen van de echte aandelen. Uit de berekening van de scheefheid volgt dat de returns allemaal een klein beetje links-scheef zijn, dit betekent dat zich net iets meer massa rechts van het

midden bevindt als je kijkt naar de kansdichtheid.

De Jarque-Bera test kijkt hoe goed de scheefheid en kurtosis overeenkomen met een normale verdeling. Omdat de bijbehorende p-waarde bij alle returns ongeveer nul is, wordt deze nulhypothese verworpen.

Een standaard manier om te kijken naar afhankelijkheid in de tijd van data is aan de hand van de autocorrelatiefunctie, deze meet hoe returns op de ene dag X_t gecorreleerd zijn met returns op een andere dag X_{t+h}. De autocorrelatiefunctie hangt af van het tijdsverschil (lag) h en wordt gegeven door

$$\rho_X(h) = \text{Cor}(X_{t+h}, X_t) = \frac{\text{Cov}(X_{t+h}, X_t)}{\text{Var(X)}} = \frac{E[(X_{t+h} - \mu)(X_t - \mu)]}{\sigma^2}$$

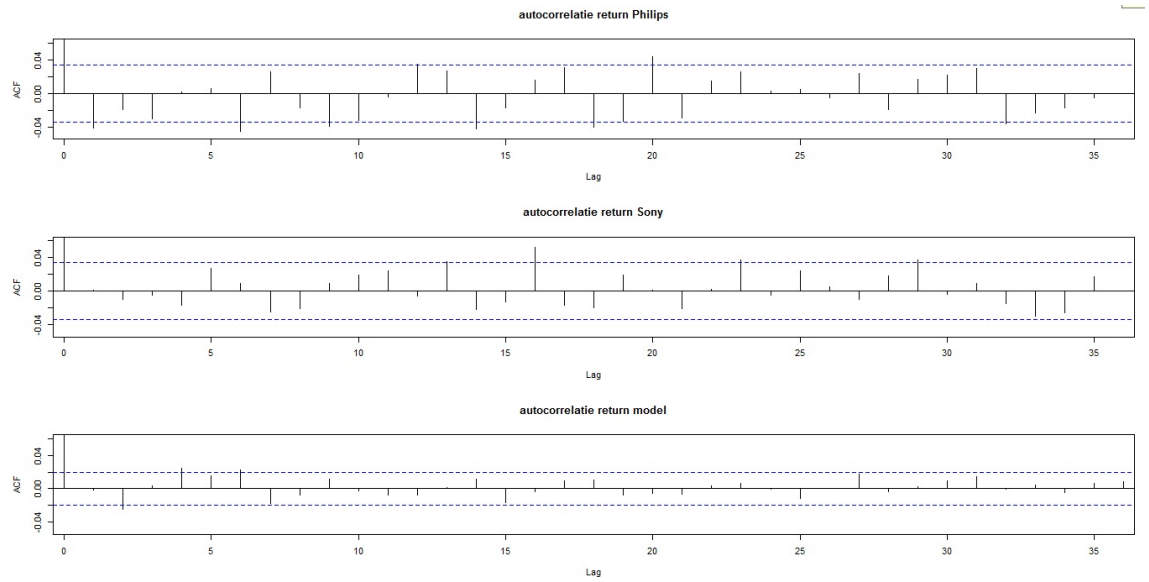
Deze formule geldt alleen als de tijdsreeks stationair is. Dat wil zeggen dat de verwachting onafhankelijk is van t en dat $\text{Cov}(X_{t+h}, X_t)$ voor elke h onafhankelijk is van t, [4]. Als deze autocorrelaties significant zijn hebben we een duidelijke aanwijzing voor afhankelijkheid.
De Ljung-Box test heeft als nulhypothese dat de data in een dataset onafhankelijk verdeeld is. Hij test dit door voor een aantal tijdsverschillen te kijken of de autocorrelatie gelijk is aan nul. Uit de kleine p-waarden voor de returns van zowel Philips, als Sony als van het model volgt dat de nulhypothese verworpen wordt en dat er dus een bepaalde afhankelijkheid in de tijd is. Voor de gekwadrateerde logreturns geldt dit nog sterker, aangezien de p-waarden daarvan nul zijn moet daar de afhankelijkheid in de tijd duidelijker zijn.

3.2 Veelvoorkomende empirische observaties

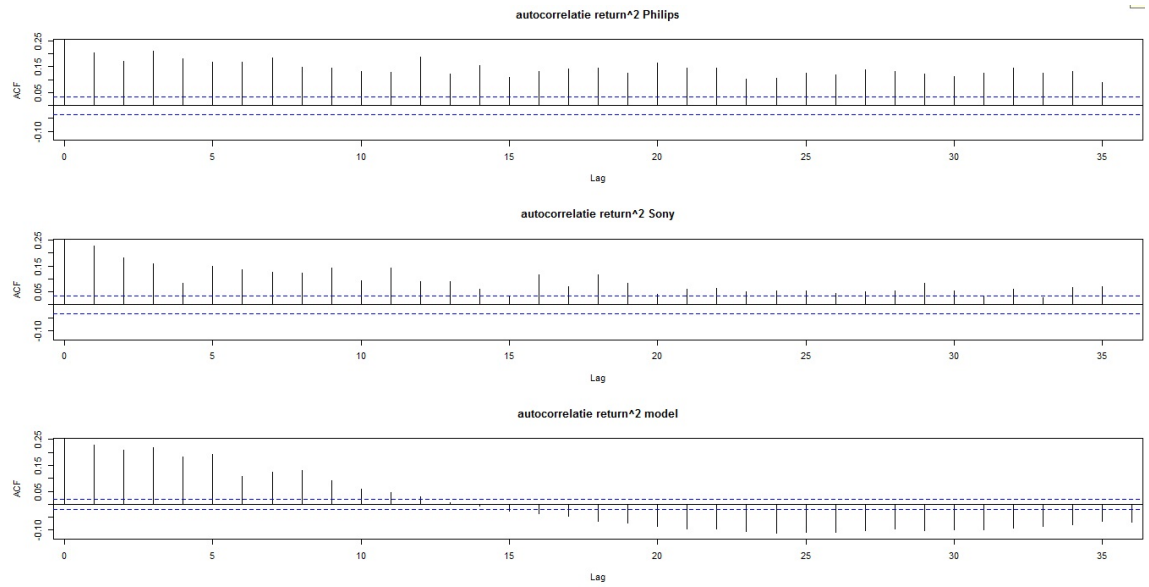
3.2.1 Volatiliteitsclusters

Met volatiliteitsclusters wordt de observatie bedoeld dat de grootte van volatiliteit de neiging heeft om samen te 'clusteren'. Dus hoge/lage volatiliteit komt vaak voor in cykels waar we een aantal dagen van hoge/lage volatiliteit achter elkaar zien. Dit zorgt ervoor dat volatiliteit deels voorspelbaar is. Als we in figuur-3.2 kijken naar de grootte van de returns zie je voor Philips en Sony wel duidelijk deze clusters terugkomen. De data die we met ons model gegenereerd hebben vertoont deze niet.

In figuur-3.3 en figuur-3.4 zijn de autocorrelaties van de logreturns en de gekwadrateerde logreturns te zien samen met een 95% betrouwbaarheidsinterval. Hieruit kunnen we aflezen dat er (zoals de Ljung-Box testen ook al aangaven) een mate van afhankelijkheid is, die zich vooral laat zien bij de gekwadrateerde logreturns.



Figuur 3.3: autocorrelatie returns



Figuur 3.4: autocorrelatie gekwadrateerde returns

3.2.2 Dikke staarten

Een verdeling met dikke staarten vertoont meer extreme uitkomsten dan een verdeling die normaal is verdeeld met dezelfde verwachting en standaarddeviatie. Een belangrijke eigenschap van financiële returns is dat de verdeling hiervan dikke staarten heeft, dus er zijn meer relatief grote en kleine returns dan we verwachtten als we een normale verdeling hadden. Er zijn twee veelgebruikte aanpakken om dikke staarten te identificeren en analyseren, beide zullen we bespreken in de komende subsecties.

Statistische toets op normaliteit

De scheefheid en kurtosis van elke normale verdeling zijn hetzelfde, namelijk 0 en 3 voor alle verdelingen. Een hogere kurtosis geeft vaak aan dat een groter deel van de variantie wordt veroorzaakt door infrequente extreme schommelingen dan voorspelt door de normale verdeling. Hoge kurtosis is dus een sterk signaal dat een verdeling dikke staarten zou kunnen hebben. Aan de hand van de waarden van de kurtosis verwachten we dus dat de verdelingen dikke staarten hebben, bij Philips en sony in sterkere mate dan bij het model, zie tabel-3.2.

	kurtosis
Philips	6,27
Sony	7,74
model	4,82

Tabel 3.2: kurtosis van de logreturns

We bekijken nu de Kolmogorov-Smirnov test, deze vergelijkt data met een referentie verdeling gebasseerd op basis van de afstanden van de twee verdelingen. Een voordeel van deze test is dat hij de hele verdeling bekijkt en gevoelig is voor zowel de positie als de vorm hiervan. Het uitvoeren van deze test voor de drie logreturns met de normale verdeling als vergelijking geeft voor allemaal een p-waarde van ongeveer nul, zie tabel-3.3. Dit geeft ook nog aan dat de kans dat ze normaal verdeeld zijn heel klein is.

	p-waarde
Philips	$1,9 * 10^{-12}$
Sony	$1,3 * 10^{-9}$
model	$2,2 * 10^{-16}$

Tabel 3.3: p-waarden van de Kolmogorov-Smirnov test

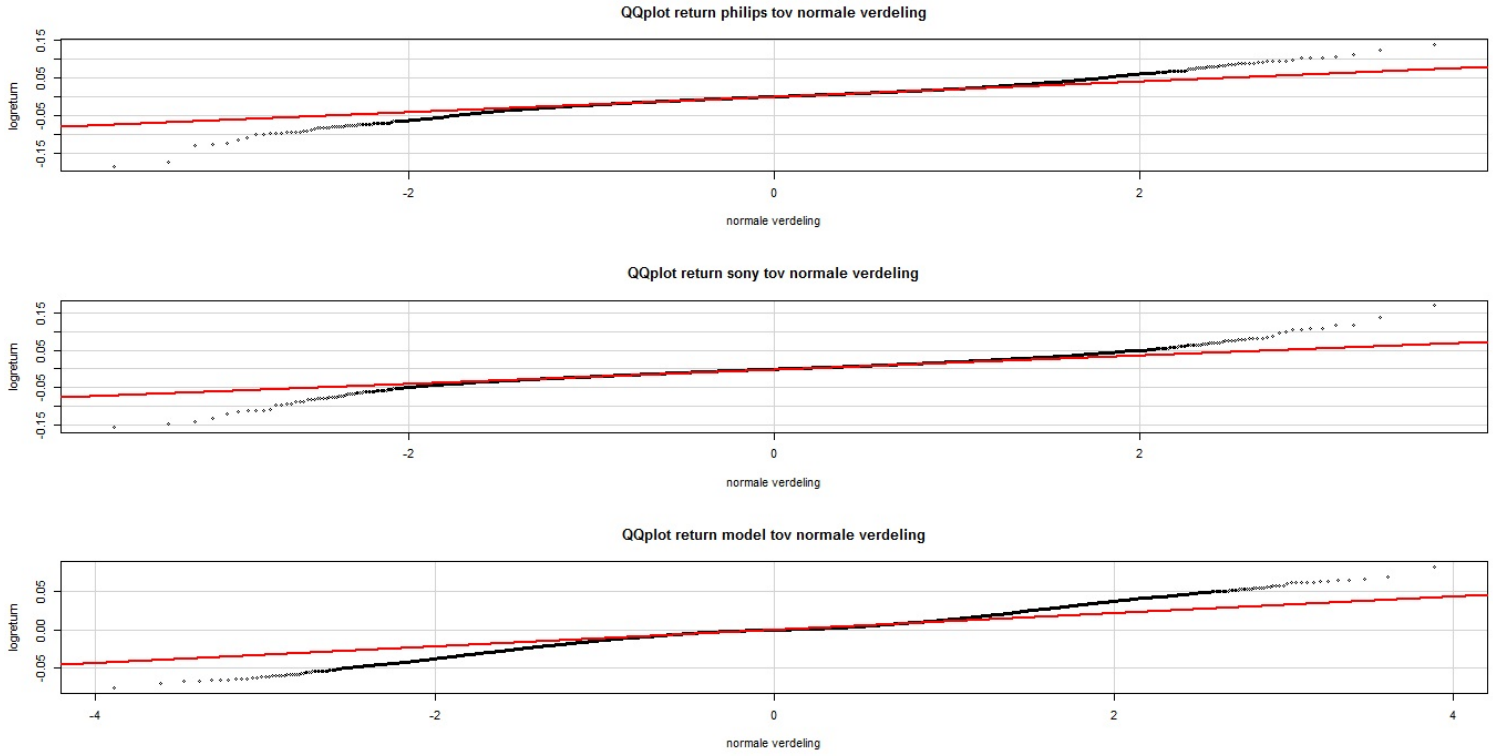
Grafische methoden

De grafische methoden die gebruikt worden om dikke staarten te ontdekken koppelen geobserveerde returns aan waarden die voorspeld zijn met een bepaalde verdeling, vaak de normale. Ze geven geen preciese beschrijving van de verdeling van de data maar kunnen wel informatie geven over de dikheid van de staarten of over hoe de verdeling verschilt van de normale.

QQ plot

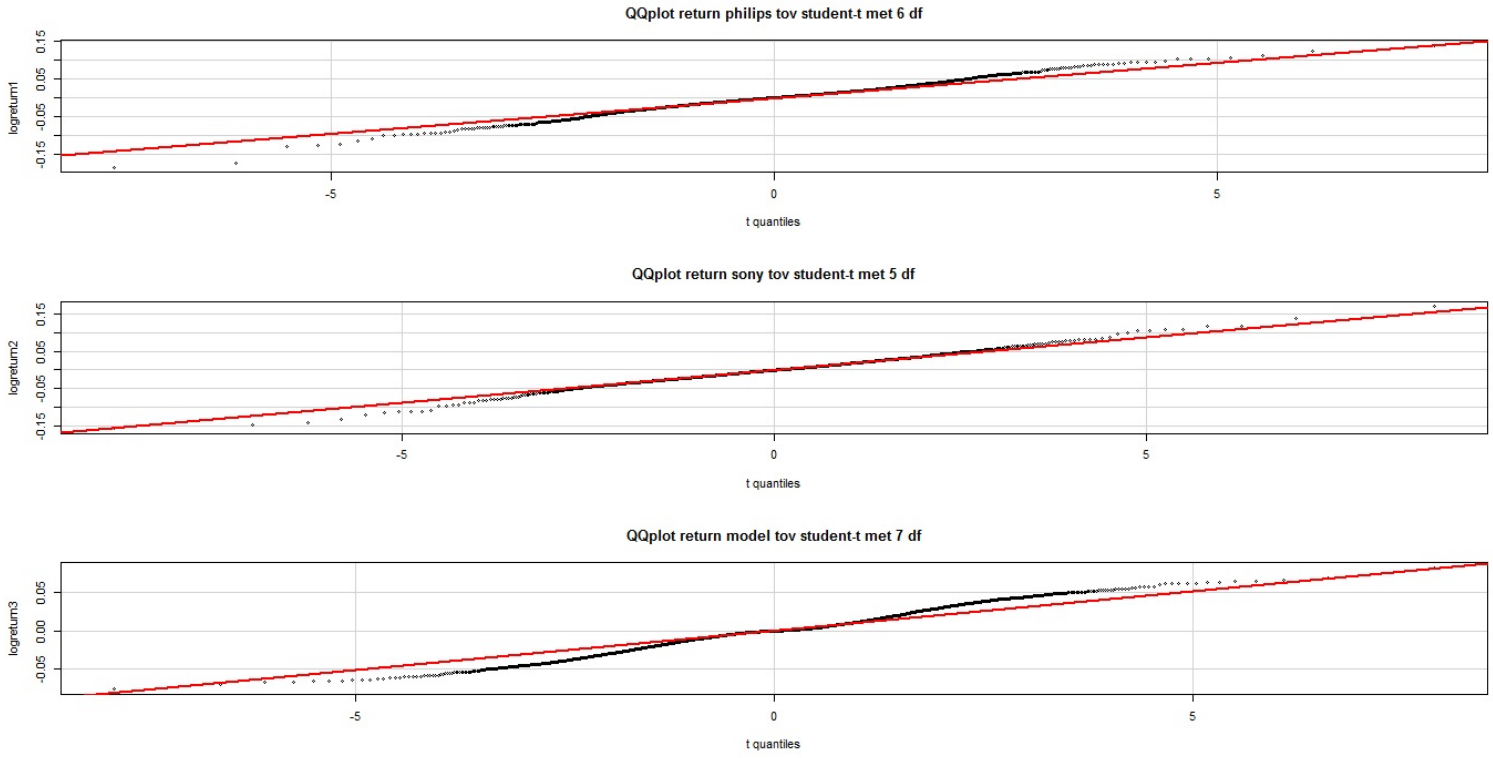
Eén van de meest voorkomende grafische methoden om staarten van een verdeling te bestuderen is een QQ plot, waarin de Q staat voor een quantile. Een bepaald p-kwantiel verdeelt je dataset in een groep kleinere waarden, fractie p en een groep grotere waarden, fractie $1 - p$. Een QQ plot vergelijkt de kwantielen van de data met de kwantielen van een referentieverdeling. Ze worden gebruikt om te kijken of twee datasets dezelfde verdeling hebben of om te kijken of een verzameling observaties een bepaalde verdeling heeft.

Als we de QQ-plots van onze returns ten opzichte van de normale verdeling bekijken, zien we dat ze significant verschillen met de normale verdeling. Aan de boven en onderkant van het plot verschillen ze erg, ze lijken dikke staarten te hebben. Zie figuur-3.5



Figuur 3.5: QQ plot logreturns tov normale verdeling

Om te analyseren hoe dik de staarten ongeveer zijn vergelijken we de logreturns met de kwantielen van een verdeling waarvan bekend is dat ze dikke staarten vertoont. Hiervoor nemen we de Student-t verdeling, hierin geeft het aantal vrijheidsgraden aan hoe dik de staarten zijn. We bekijken de QQ plots van de logreturns ten opzichte van Student-t verdelingen met steeds een vrijheidsgraad groter of kleiner. We concluderen dat de staart van de verdeling van de logreturns van Philips ongeveer overeenkomt met die van een t(6) verdeling, die van Sony met die van een t(5) verdeling en die van het model met die van een t(7) verdeling. Zie figuur-3.6 voor deze QQ plots.



Figuur 3.6: QQ plot logreturns tov de verschillende Student-t verdelingen

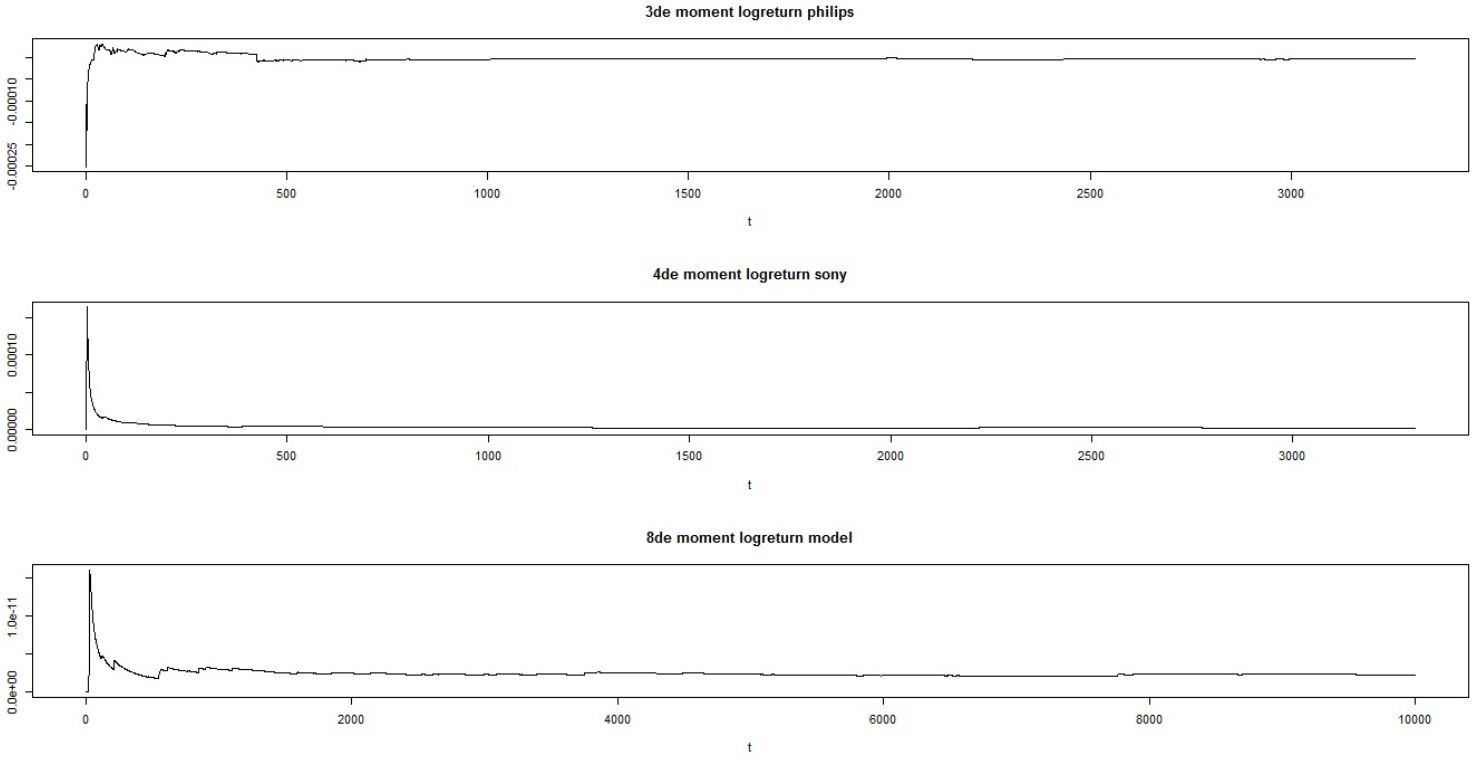
Momenten
De dikheid van een staart van een verdeling wordt gemeten door de staartindex ι. Er geldt dat hoe dikker de staart is, hoe lager de index. In het geval van de Student-t verdeling komt ι overeen met het aantal vrijheidsgraden. We bekijken nu het m-de cerntrale moment van een kansverdeling, deze wordt gegeven door:

$$E[(X-\mu)^m] = \int_{-\infty}^{\infty} (x-\mu)^m f(x)dx$$

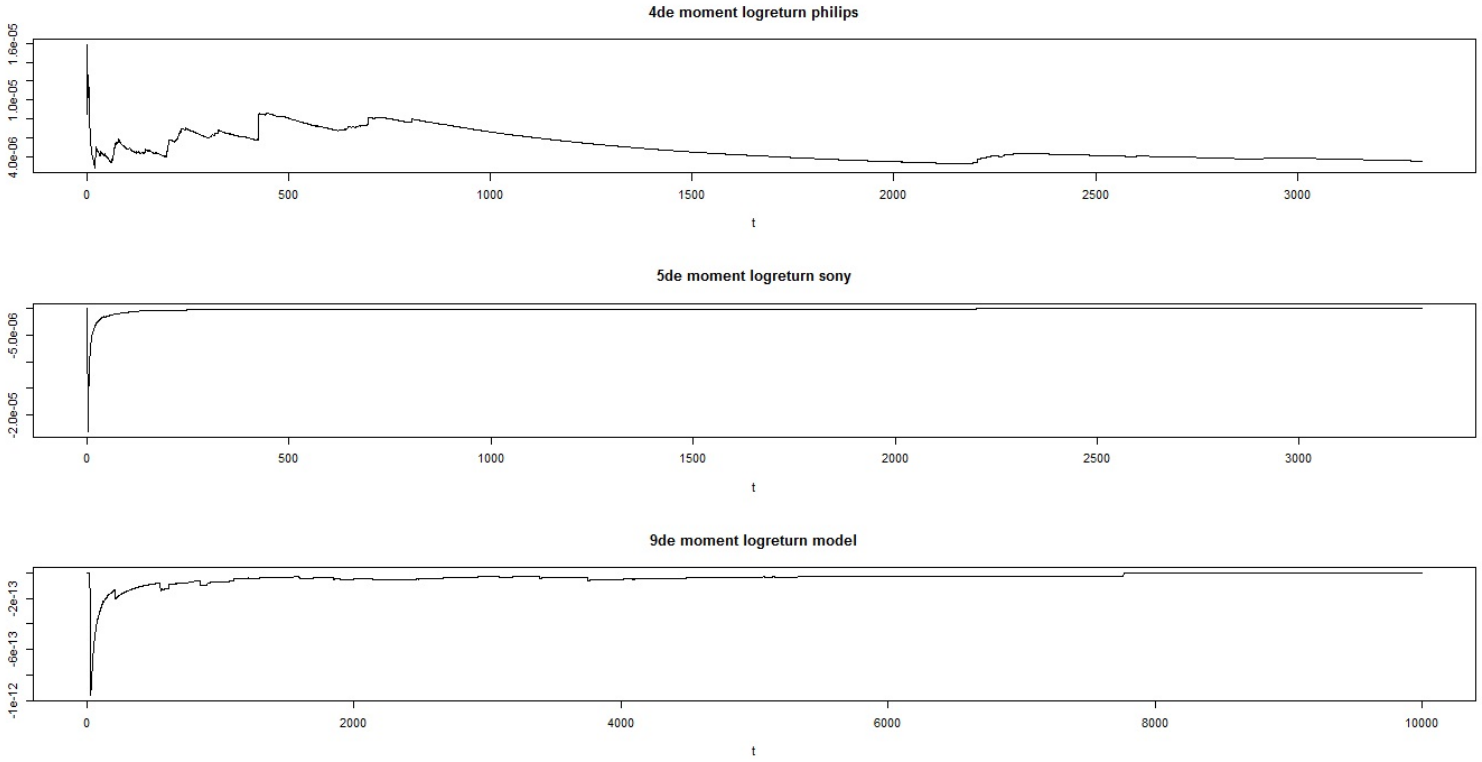
Als een verdeling dikke staarten heeft dan kunnen we deze momenten alleen berekenen voor $m < \iota$. Dit impliceert dat als het aantal begrensde momenten eindig is, de verdeling wel dikke staarten moet hebben. We kunnen nu dus de dikheid van een staart van een steekproef $(x_1, ..., x_t)$ analyseren door het m-de moment van de steekproef te plotten voor elke t.

$$\frac{1}{t}\sum_{i=1}^{t} x_i^m$$

Als hierbij geldt $m < \iota$ dan bestaat het m-de moment, dus moet deze grafiek convergeren. En als $m > \iota$ dan convergeert de grafiek van het m-de moment niet. Hiermee kunnen we de dikheid van de staart afschatten. Voor Philips komen we met deze methode ongeveer uit op ι tussen 3 en 4, voor Sony tussen 4 en 5 en voor het model tussen 8 en 9. In figuur-3.7 staan de momenten waarvoor de grafieken nog net convergeren, in figuur-3.8 die waarvoor de grafieken niet meer convergent zijn.

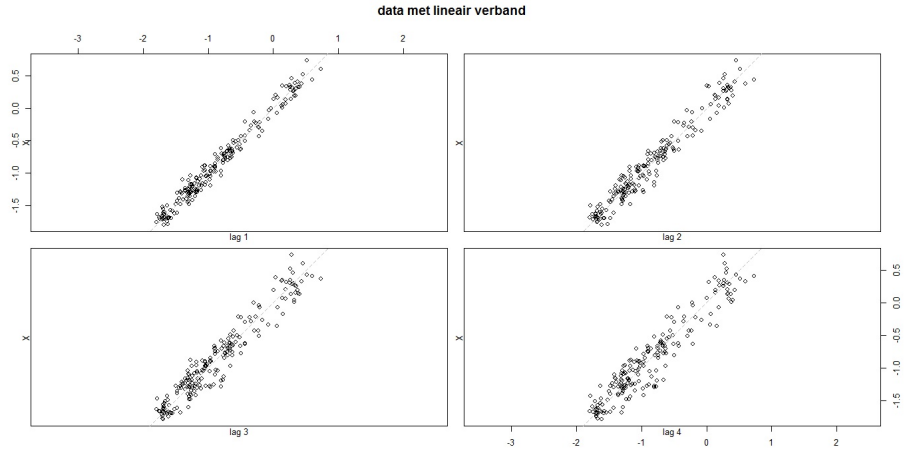


Figuur 3.7: opgetelde momenten ondergrens



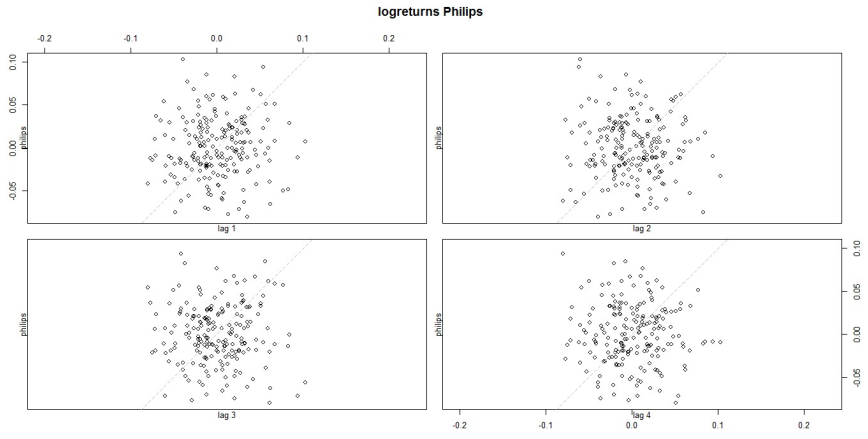
Figuur 3.8: opgetelde momenten bovengrens

3.2.3 Niet-lineaire afhankelijkheid

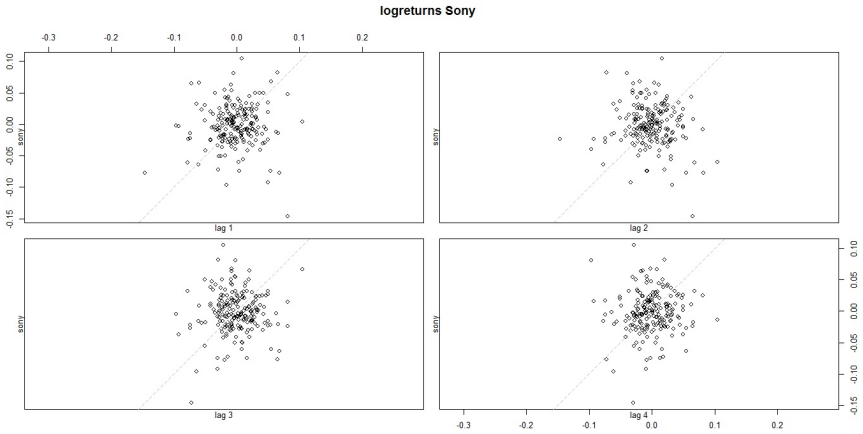


Figuur 3.9: lagplot van een lineair afhankelijke tijdreeks

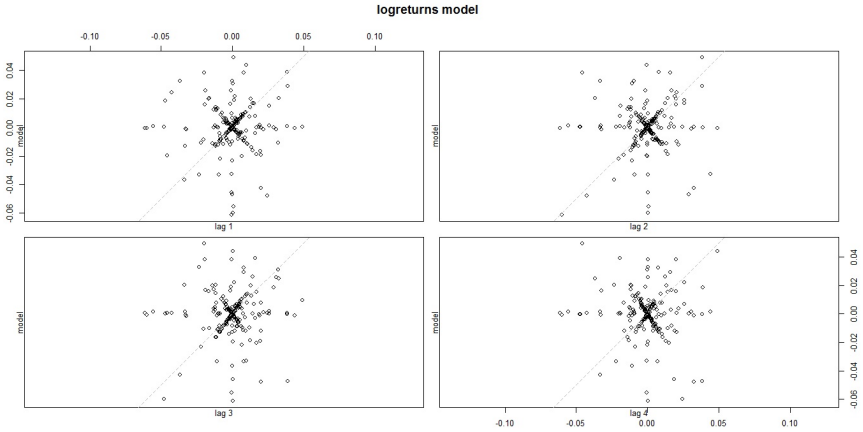
Sommige statistische modellen om beurskoerzen te beschrijven nemen aan dat er een bepaalde lineaire afhankelijkheid in de tijd bestaat. Om dit te bestuderen bekijken we lagplots, deze zetten X_t uit tegen X_{t-n} voor tijdsverschil (lag) n, met $n = 1, 2, ...$ Zie figuur-3.9 voor een lagplot van data met een lineaire afhankelijkheid in de tijd, de datapunten liggen dan rond de lijn $y = x$. In figuur-3.10, figuur-3.11 en figuur-3.12 is te zien dat zowel de logreturns van Sony en Philips als de logreturns gegenereerd met het model geen lineaire afhankelijkheid in de tijd hebben. Wel is te zien in figuur-3.12 dat de logreturns gegenereerd met het model toch een bepaald patroon vertonen in de lagplots. Het is aannemelijk dat dit te maken heeft met de regelmatigheid in de logreturns die we ook al zagen in figuur-3.2, helaas is het niet gelukt dit verder te verklaren.



Figuur 3.10: lagplot van de logreturns van Philips



Figuur 3.11: lagplot van de logreturns van Sony



Figuur 3.12: lagplot van de logreturns gegenereert met het model

Hoofdstuk 4

Parameters schatten met het ABC-algoritme

Nu hebben we een dataset, dit kan bijvoorbeeld een aandelenkoers zijn. We willen weten welke waarden we de parameters in een bepaald model moeten geven om deze dataset het best te simuleren. In de komende hoofdstukken bekijken we dit probleem door een dataset te nemen die gegenereerd is door een model. Vervolgens schatten we voor eéń van de parameters van dat model welke waarde deze moet hebben gehad bij de simulatie van de dataset. De rest van de parameters veronderstellen we bekend.

4.1 Methode

Veel schattingsmethoden zijn gebasseerd op de likelihoodfunctie. Dit is de kans op de geobserveerde data $x_1, ..., x_n$ als functie van de parameter θ die geschat moet worden,

$$\text{lik}(\theta) = f(x_1, ..., x_n|\theta)$$

met f de kansdichtheidsfunctie. Neem bijvoorbeeld de maximum likelihood methode waarbij deze functie gemaximaliseerd wordt. Een andere manier is de Bayesiaanse aanpak voor het schatten van parameters, hierin wordt de parameter beschouwd als een stochastische variabele. Deze heeft een a priori verdeling $f(\theta)$ die representeert wat al bekend is van de parameter voordat de geobserveerde data bekeken is. De verdeling van θ gegeven de geobserveerde data, de a postriori verdeling, wordt dan gegeven door [5]

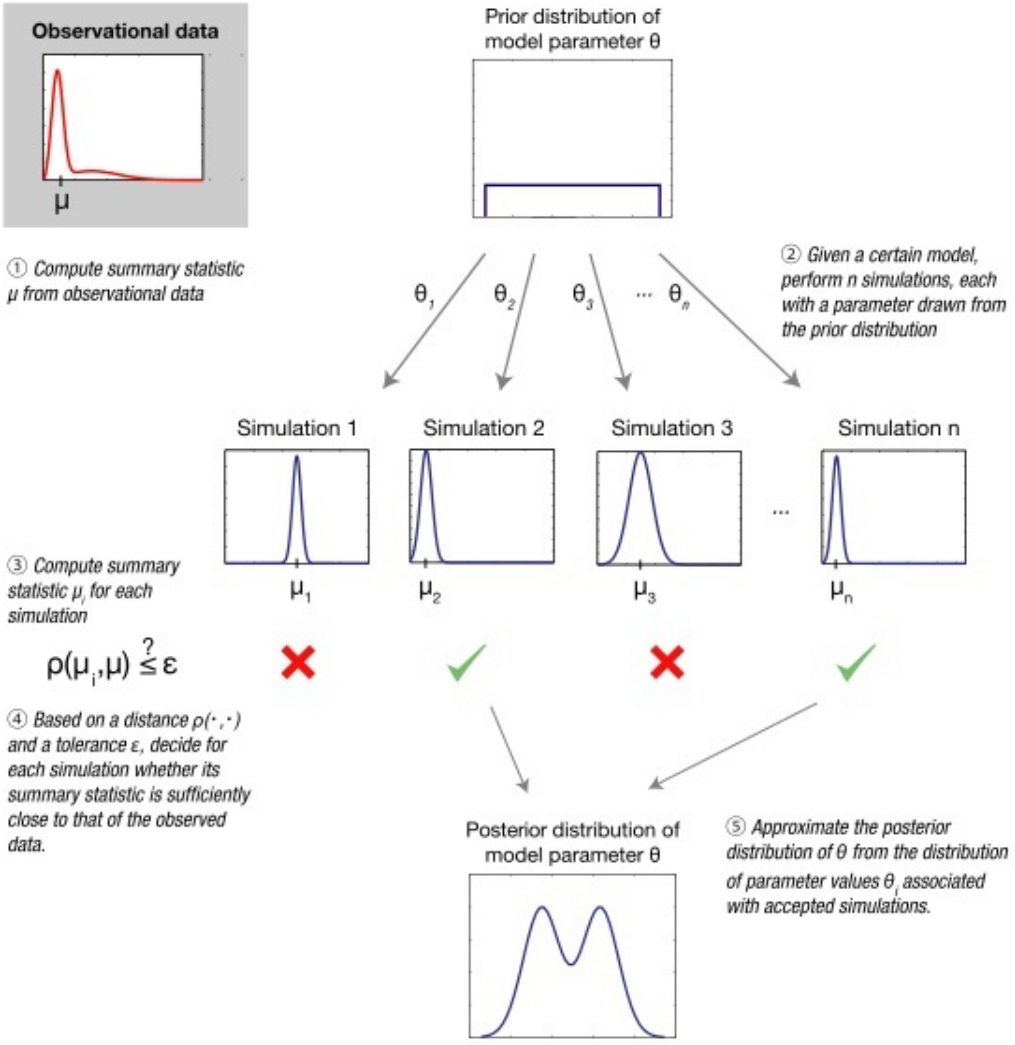
$$f(\theta|x_1, ..., x_n) = \frac{f(x_1, ..., x_n|\theta)f(\theta)}{\int f(x_1, ..., x_n|\theta)f(\theta)d\theta}$$

Nu is het model waar we parameters voor willen schatten te ingewikkeld om een likelihoodfunctie op te stellen. Er wordt dan ook een methode gebruikt die hier een oplossing voor heeft, een zogenaamde Approximate Bayesian Computational (ABC) methode [6]. Deze methode heeft zijn oosprong in de Bayesiaanse statistiek en beoogt de a postriori verdeling voor θ te benaderen zonder de likelihoodfunctie te gebruiken. In dit hoofdstuk bekijken we allereerst de meest simpele ABC-methode.

In essentie wordt een bepaalde θ uit de a priori verdeling getrokken en deze wordt gebruikt om volgens het model data te simuleren, deze data wordt vergeleken met de geobseveerde data. Als de datasets genoeg overeen komen wordt de θ geaccepteerd, anders wordt deze verworpen. Met de geaccepteerde θ's wordt de a postriori verdeling benaderd. Dit gaat als volgt:

1. Kies een samenvattingsvariabele μ die de geobserveerde data beschrijft (bijvoorbeeld het gemiddelde) en bereken deze voor de geobserveerde data.
2. Trek n keer een θ uit de a priori verdeling en simuleer hiermee n keer het model.
3. Bereken voor de data uit elke simulatie de samenvattingsvariabele μ_i.
4. Kies een tolerantiegrens ϵ en een maat ρ om te bepalen of de data genoeg overeenkomt met de geobserveerde data (bijvoorbeeld de afstand tussen de samenvattingsvariabelen).
5. Accepteer elke θ_i met $\rho(\mu, \mu_i) \leq \epsilon$.
6. Benader de a postriori verdeling van θ met de verdeling van de θ_i's die horen bij de geaccepteerde simulaties.

Zie figuur-4.1.



Figuur 4.1: standaard ABC-methode, bron:Wikipedia [7]

4.2 Eenvoudige illustratie van het ABC-algoritme

Om deze methode te testen en nader te bekijken is hij eerst toegepast op een simpel model waarvoor de parameter die geschat wordt bekend is. We genereren data uit het model

$$X_t = 0.8X_{t-1} + z_t, \text{ met } X_0 = 0, \ z_t \sim N(0, \sigma^2), \ t = 0, ..., M$$

Hierbij wordt verondersteld dat $z_1, ..., z_M$ onafhankelijk zijn. Stel nu dat het model $X_t = \beta X_{t-1} + z_t$ bekend is, maar $\beta_0 = 0,8$ onbekend is, die wordt geschat met de methode.

Als samenvattingsvariabele wordt de som van de autocorrelaties van de eerste vijf tijdsverschillen van de data gekozen. Als maat om te bepalen of de data goed genoeg overeenkomt met de geobserveerde data wordt de Euclidische afstand tussen de samenvattingsvariabelen genomen. Er worden Q simulaties uitgevoerd en de tolerantiegrens ϵ wordt zo gekozen dat de beste $0,01\%$ van de β's in deze simulaties geaccepteerd worden. Als a priori verdeling van β nemen we de uniforme verdeling op $[-1,1]$.

4.2.1 Exacte berekening van de a posteriori verdeling

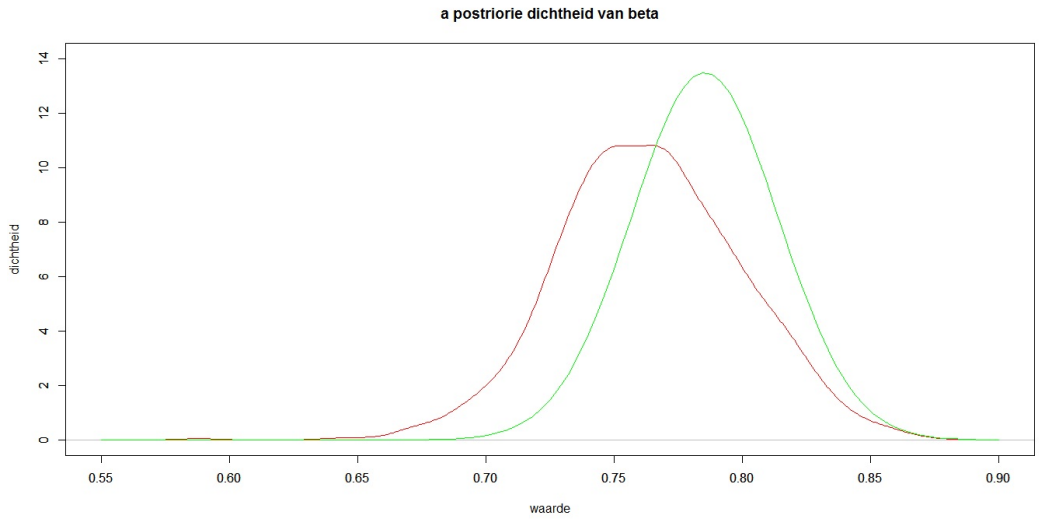
Door het eenvoudige karakter van dit model is het mogelijk om in dit geval exact de a postriori verdeling te bepalen. Bekend is $X_t = \beta X_{t-1} + z_t$ en $z_t \sim N(0,\sigma^2)$ met σ^2 bekend. Hieruit volgt dat $z_t = X_t - \beta X_{t-1} \sim N(0,\sigma^2)$. De likelihoodfunctie is nu

$$f(x_1,...,x_M|\beta) = f(x_1)\prod_{t=2}^{M}\frac{1}{\sqrt{2\pi\sigma^2}}\exp^{-\frac{1}{2\sigma^2}(x_t-\beta x_{t-1})^2}$$

Voor lange tijdreeksen mag $f(x_1)$ verwaarloosd worden, deze wordt dan ook weggelaten bij de berekening. De uniforme verdeling op $[-1,1]$ wordt als a priori verdeling genomen, $f(\beta) = 1_{[-1,1]}(\beta)$. De a postriori verdeling is nu

$$\begin{aligned} f(\beta|x_1,...,x_M) &= \frac{f(x_1,...,x_M|\beta)f(\beta)}{\int f(x_1,...,x_M|\beta)f(\beta)d\beta} \\ &= \frac{\prod_{t=2}^{M}\frac{1}{\sqrt{2\pi\sigma^2}}\exp^{-\frac{1}{2\sigma^2}(x_t-\beta x_{t-1})^2}1_{[-1,1]}(\beta)}{\int_{-1}^{1}\prod_{t=2}^{M}\frac{1}{\sqrt{2\pi\sigma^2}}\exp^{-\frac{1}{2\sigma^2}(x_t-\beta x_{t-1})^2}d\beta} \\ &= \frac{\exp^{\frac{1}{2\sigma^2}\sum_{t=2}^{M}(x_t-\beta x_{t-1})^2}1_{[-1,1]}(\beta)}{\int_{-1}^{1}\exp^{\frac{1}{2\sigma^2}\sum_{t=2}^{M}(x_t-\beta x_{t-1})^2}d\beta} \end{aligned}$$

4.2.2 Resultaten

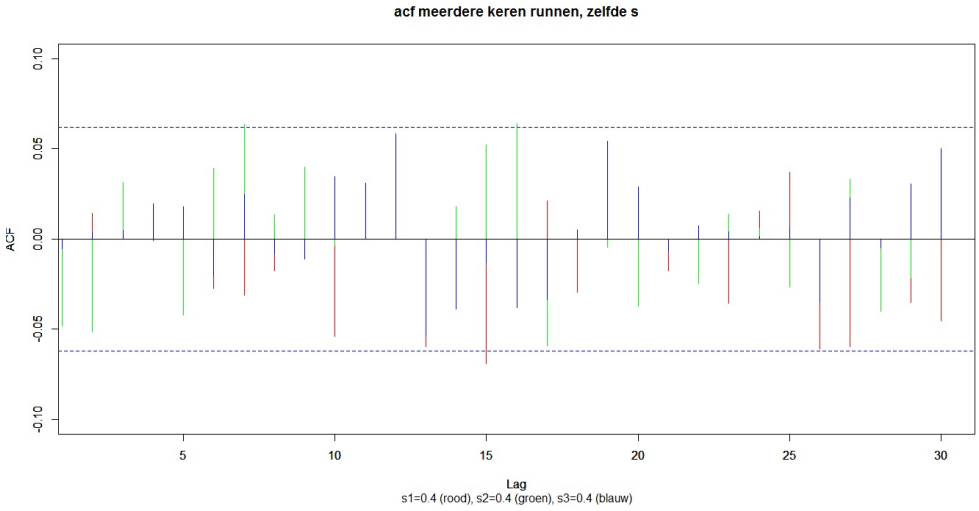


Figuur 4.2: ABC benadering van de a postriori verdeling (rood) en de numerieke benadering van de exacte a posteriori verdeling (groen)

Een test met $Q = 10^6$ simulaties en datasets van lengte $M = 500$ geeft als schatting voor $\hat{\beta}_0 = 0.76$, hierbij is $\sigma = 0.01$ gekozen. Zie figuur-4.2 voor de dichtheid van de 0.01% beste β's. In de figuur is ook de exacte a postriori verdeling te zien, deze is nummeriek benaderd. Hoewel de rekentijd relatief lang is (89 minuten), gaf de schatting wel een goede benadering voor β_0, hij zit er met 0.76 maar 2% naast. De ABC benadering van de a posteriori verdeling lijkt op de nummerieke benadering van de exacte a postriori verdeling maar heeft de piek minder dicht bij 0.8. De methode lijkt goed te werken, al blijft er ruimte voor verbetering.

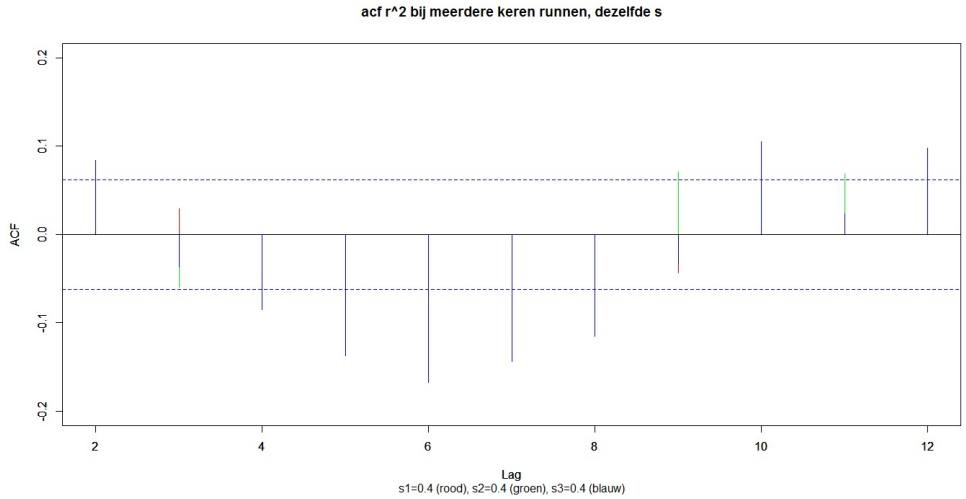
4.3 ABC-algoritme toegepast op het model

Met het model (§2.1) wordt er data (r, de logreturns) gegenereerd met een bepaalde s_0, alle andere parameters worden bekend verondersteld, dit is de geobserveerde data. Vervolgens willen we met de ABC-methode schatten met welke s deze data gegenereerd is.
Als samenvatttingsvariabele werd de som van de autocorrelaties van de logreturn van de eerste 10 tijdsverschillen genomen. Deze samenvattingsvariabele bleek de data niet goed genoeg samen te vatten. Dit bleek uit het feit dat meerdere simulaties van het model met dezelfde s nog steeds zorgde voor verschillende autocorrelatiefuncties, zie figuur-4.3.

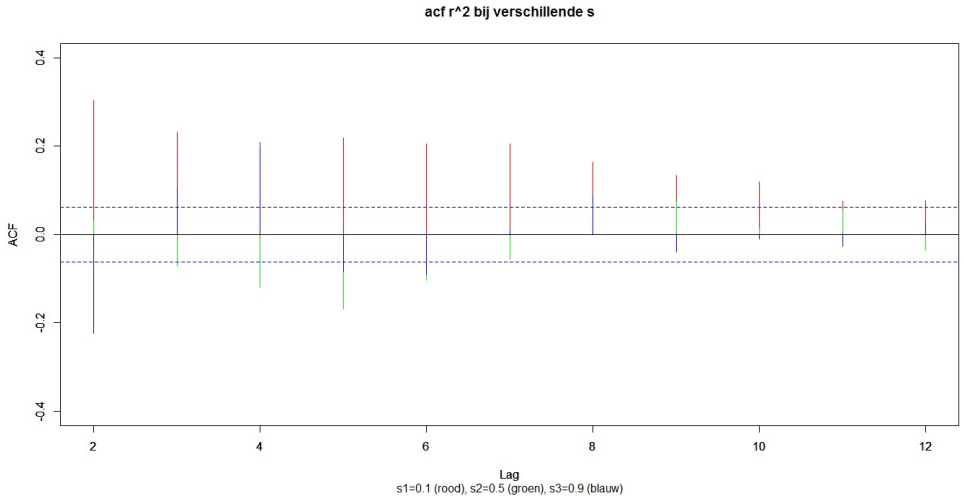


Figuur 4.3: autocorrelatiefunctie van meerdere simulaties met dezelfde s

De autocorrelaties van de logreturns in het kwadraat bleek wel te werken. Voor de eerste tien tijdsverschillen blijkt daarvoor de autocorrelatiefuncties wel goed overeen te komen voor simulaties met dezelfde s, de lijnen liggen bijna precies over elkaar heen, zie figuur-4.4. Maar ook verschillen de autocorrelatiefuncties voor simulaties met verschillende s, zie figuur-4.5.



Figuur 4.4: autocorrelatiefunctie van de logreturn in het kwadraat van meerdere simulaties met dezelfde s



Figuur 4.5: autocorrelatiefunctie van de logreturn in het kwadraat van meerdere simulaties met verschillende s

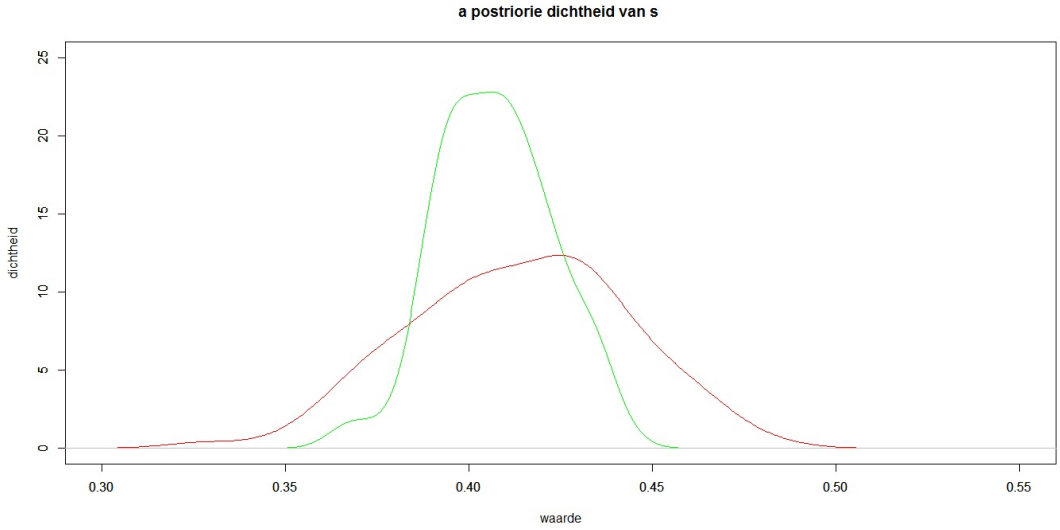
Als maat om te bepalen of de data genoeg overeenkomt met de geobserveerde data nemen we nu $\sum_{i=2}^{10} |$verschil in autocorrelatie van r^2 op tijdsverschil i$|$.
Als a priori verdeling van s kiezen we de uniforme verdeling op $[0, 1]$ aangezien s een kans is. Er worden Q simulaties uitgevoerd en de tolerantiegrens ϵ wordt zo gekozen dat de beste 10% of de beste 1% van de kandidaten voor s wordt geaccepteerd.

4.3.1 Resultaten

Als parameters voor het model (zie §2.1) kiezen we $T = 1000$, $N = 500$, $D = 0.001$, $\lambda = 10$, $g(z) = \frac{z}{\lambda}$ en $\theta_i(0) = \theta_0 = \frac{1}{2}$. We simuleren de geobserveerde data met $s_0 = 0.4$.
Het algoritme met $Q = 1000$ simulaties geeft als schatting voor $\hat{s_0} = 0.413$, de rekentijd hiervoor is ongeveer 78 minuten. Het algoritme met $Q = 10000$ simulaties geeft als schatting voor $\hat{s_0} = 0.407$, de rekentijd is lang: bijna 16 uur, maar de schatting is preciezer. Zie figuur-4.6 voor de ABC-schatting van de a postriori dichtheid van s voor beide schattingen.

Q	$\hat{\mathbf{s_0}}$	**rekentijd (uur)**
1000	0.413	1.3
10000	0.407	16

Tabel 4.1: resultaten ABC methode op model



Figuur 4.6: ABC benadering van de a postriori verdeling van s, Q = 1000 (rood) en Q = 10000 (groen)

Hoofdstuk 5

Eerste uitbreiding op het ABC-algoritme

Het is inefficient om voor elke nieuwe simulatie kandidaten te trekken uit de a priori verdeling als al bekend is dat de voorgaande kandidaat geaccepteerd is. In dat geval wil je daar in de buurt blijven. Dit idee wordt gebruikt bij de volgende verbetering op het ABC-algoritme. We bekijken een Markov Chain Monte Carlo (MCMC) ABC-algoritme [6] dat een Markovketen construeert die bij benadering de a postriori verdeling als stationaire verdeling heeft.

5.1 Markovketens

Een Markovketen beschrijft een systeem dat zich door een aantal toestanden beweegt en dat stapsgewijs van de ene naar de andere toestand overgaat. We noemen X_n de toestand op tijdstip n. We veronderstellen dat alle toestanden in $\mathbb{R}^d$ zitten. De transitiekern $\mathrm{P}(x, A)$ geeft de kans dat het proces zich op een bepaald tijdstip n in de verzameling $A \in \mathbb{R}^d$ bevindt, gegeven dat het zich op tijdstip $n-1$ in x bevond. Ofwel de kans om van x naar een punt in de verzameling A te gaan [8]. Het is mogelijk dat het systeem in x blijft, dus $\mathrm{P}(x, \{x\})$ is niet persé nul. Het toekomstige gedrag van het systeem is volledig te beschrijven als bekend is in welke toestand het systeem zich bevindt, het is niet relevant hoe het systeem in de huidige toestand is gekomen. We hebben dus $P(X_{n+1} \in A | X_n = x) \stackrel{def}{=\!=} \mathrm{P}(x, A)\ \forall n$. Een iteratie is een tijdstap, waarbij de volgende toestand volgens de transitiekern gekozen wordt. Onder bepaalde voorwaarden convergeert de Markovketen naar een invariante verdeling voor $n \to \infty$.

Stel de toestand op tijdstip n heeft verdeling π, $X_n \sim \pi$. We bekijken nu de kans dat de toestand op tijdstip $n+1$ in het interval (a,b) ligt:

$$\begin{aligned} P(X_{n+1} \in (a,b)) &= \int_{x \in \mathbb{R}^d} P(X_{n+1} \in (a,b) | X_n = x) \pi(x) dx \\ &= \int_{\mathbb{R}^d} \int_{y=a}^{b} \mathrm{P}(x, dy)\ \pi(x) dx \overset{(d.b.)}{=\!=\!=} \int_{y=a}^{b} \int_{\mathbb{R}^d} \mathrm{P}(y, dx) \pi(y) dy \\ &= \int_{y=a}^{b} \pi(y) dy = \pi((a,b)) = P(X_n \in (a,b)) \end{aligned} \tag{5.1}$$

Hierbij volgt de eerste gelijkheid uit voorwaardelijke kansen [5],
$f(y) = \int f(y|x) f(x) dx$. Voor de gelijkheid (d.b.) moet de stelling van Fubini [9] gelden. Deze stelling zegt dat we de integratievolgorde mogen omdraaien als $\int_{\mathbb{R}^d \times (a,b)} |\mathrm{P}(x, dy)\ \pi(x)| dx$ eindig is. Ook is de gelijkheid (d.b.) alleen is geldig als voldaan is aan de "detailed balance" vergelijking. Deze zegt intuitief dat de kans om van x naar y te gaan even groot moet zijn als de kans om van y naar x te gaan. De verdeling van X_n blijft dan hetzelfde en we hebben dus een invariante verdeling. De invariante verdeling voldoet dus aan

$$\pi(y) dy = \int_{\mathbb{R}^d} \mathrm{P}(x, dy) \pi(x) dx \tag{5.2}$$

Bij MCMC methoden construeren we $\mathrm{P}(x, dy)$ zodanig dat π gelijk is aan de a postriori verdeling. Om $\mathrm{P}(x, dy)$ te vinden nemen we aan dat deze voor een functie $p(x,y)$ te schrijven is als

$$\mathrm{P}(x, dy) = p(x,y) dy + r(x) \delta_x(dy) \tag{5.3}$$

met $p(x,x) = 0$,$\delta_x(dy) = \begin{cases} 1 & \text{als } x \in dy \\ 0 & \text{anders} \end{cases}$, en $r(x) = 1 - \int_{\mathbb{R}^d} p(x,y) dy$ de kans dat je in x blijft. Als deze $p(x,y)$ voldoet aan de "detailed balance" vergelijking

$$\pi(x) p(x,y) = \pi(y) p(y,x) \tag{5.4}$$

dan is $\pi(\cdot)$ de invariante verdeling waar $\mathrm{P}(x, \cdot)$ naar convergeert [8]. Om dit te laten zien gebruiken we (5.2),(5.3),(5.4) en de stelling van Fubini:

$$\begin{aligned} \int \mathrm{P}(x, A) \pi(x) dx &= \int \left[\int_A p(x,y) dy \right] \pi(x) dx + \int r(x) \delta_x(A) \pi(x) dx \\ &= \int_A \left[\int p(x,y) \pi(x) dx \right] dy + \int_A r(x) \pi(x) dx = \int_A \left[\int p(y,x) \pi(y) dx \right] dy \\ &\quad + \int_A r(x) \pi(x) dx = \int_A (1 - r(y)) \pi(y) dy + \int_A r(x) \pi(x) dx = \int_A \pi(y) dy \end{aligned}$$

Nu zoeken we een $p(x,y)$ die aan (5.4) voldoet met de Metropolis-Hastings methode. Stel dat we een dichtheid hebben die kandidaten genereerd $q(x,y)$ met $\int q(x,y)dy = 1$, als je in toestand x bent dan genereerd deze dichtheid y uit $q(x,y)$. Als $q(x,y)$ zelf voldoet aan (5.4) $\forall x,y$ dan zijn we klaar. Meestal is dit niet zo, er geldt bijvoorbeeld

$$\pi(x)q(x,y) > \pi(y)q(y,x) \tag{5.5}$$

het proces gaat te vaak van x naar y en te weinig van y naar x. We willen nu het aantal bewegingen van x naar y verminderen door de kans $\alpha(x,y)$ dat je van x naar y gaat te introduceren. Bewegingen van x naar y worden gemaakt volgens $p_{MH}(x,y) \equiv q(x,y)\alpha(x,y)$, $x \neq y$. Omdat het proces te weinig van y naar x gaat maken we $\alpha(y,x)$ zo groot mogelijk, dus $\alpha(y,x) = 1$. Nu wordt $\alpha(x,y)$ bepaald door het feit dat p_{MH} moet voldoen aan (5.4). Er moet gelden

$$\pi(x)q(x,y)\alpha(x,y) = \pi(y)q(y,x)\alpha(y,x) = \pi(y)q(y,x)$$

dus $\alpha(x,y) = \frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}$. Als (5.5) omgedraaid is nemen we $\alpha(x,y) = 1$ en leiden we $\alpha(y,x)$ af als hierboven.

Dus om ervoor te zorgen dat $p_{MH}(x,y)$ voldoet aan (5.4) hebben we nu de acceptatiekans

$$\alpha(x,y) = \min\left(\frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}, 1\right)$$

Als laatste bekijken we de kans dat het proces in x blijft, dit is nu $r(x) = 1 - \int_{\mathbb{R}^d} q(x,y)\alpha(x,y)dy$. Dus de transitiekern van de Metropolis-Hastings keten wordt gegeven door

$$\mathrm{P}_{MH}(x,dy) = q(x,y)\alpha(x,y)dy + \left[1 - \int_{\mathbb{R}^d} q(x,y)\alpha(x,y)dy)\right]\delta_x(dy)$$

Omdat $p_{MH}(x,y)$ door zijn constructie aan (5.4) voldoet, heeft deze transitiekern $\pi(\cdot)$ als invariante verdeling.

We krijgen nu het M-H algoritme als volgt:

- herhaal voor $i = 1, 2, .., N$
- begin
  - genereer y uit $q(x_i, \cdot)$
  - genereer $u \sim U(0,1)$
  - als $u \leq \alpha(x_i, y) \rightarrow$ neem $x_{i+1} = y$
  - anders $\rightarrow$ neem $x_{i+1} = x_i$
- eind

5.2 MCMC ABC-algoritme

Nu hebben we geobserveerde data, $w = (W_1, .., W_T)$, en we willen de a postriori verdeling $\pi(\theta|w)$ benaderen. Deze benadering van de a postriori verdeling is afhankelijk van de tolerantiegrens ϵ bij ABC-methoden, we zoeken $\pi_\epsilon(\theta|w)$. Dit doen we door een Markovketen te construeren met als invariante verdeling de de benadering van de a postriori verdeling, $\pi_\epsilon(\theta, z|w)$. Hierin is $z = (Z_1, .., Z_T)$ de gegenereerde data volgens het model f. We weten [6]

$$\pi_\epsilon(\theta, z|w) = \frac{f(z|\theta)\pi(\theta)I_{A_{\epsilon,w}}(z)}{\int\limits_{A_{\epsilon,w}\times\theta} f(z|\theta)\pi(\theta)dzd\theta} \propto f(z|\theta)\pi(\theta)I_{A_{\epsilon,w}}(z) \tag{5.6}$$

met $A_{\epsilon,w}$ de verzameling van alle z gegenereert met een θ die voor deze ϵ en deze geobserveerde data w geaccepteerd wordt. Dus $A_{\epsilon,w} = \{z : \rho(\mu(z), \mu(w)) \leq \epsilon\}$, met ρ de afstand en μ de samenvattingsmaat, zie §4.1. Merk op
$\int \pi_\epsilon(\theta, z|w)dz \propto \pi(\theta) \underbrace{\int\limits_{A_{\epsilon,w}\times\theta} f(z|\theta)dz}_{f(w|\theta) \text{ als } \epsilon\to 0}$, dus $\int \pi_\epsilon(\theta, z|w)dz \to \pi(\theta)f(w|\theta) \propto \pi(\theta|w)$
als $\epsilon \to 0$. Dus als ϵ klein is dan heeft $\pi_\epsilon(\theta, z|w)$ ongeveer de juiste marginale verdeling [6].

We bekijken nu de acceptatiekans $\alpha(x, y) = \min\left(\frac{\pi(y)}{\pi(x)}\frac{q(y,x)}{q(x,y)}, 1\right)$. Hierin nemen we:

- $x = (\theta^{(t-1)}, z^{(t-1)})$: θ en de bijbehorende gegenereerde data z op de vorige tijdstap
- $y = (\theta', z')$: de nieuwe kandidaten hiervoor
- $\pi(\cdot) = \pi_\epsilon(\cdot|w)$: de invariante verdeling
- $q(x, y) = K(\theta'|\theta^{(t-1)})f(z'|\theta')$: Om van x naar y te komen met behulp van het voorstel $q(x, y)$ ga je eerst van $\theta^{(t-1)}$ naar θ' door θ' te genereren met $K(\theta'|\theta^{(t-1)})$. Vervolgens genereer je de data z' met behulp van het model (we beschrijven dit met f), dus volgens de likelihoodfunctie $f(z'|\theta')$

We krijgen:

$$\alpha(x, y) = \frac{\pi(y)}{\pi(x)}\frac{q(y, x)}{q(x, y)} = \frac{\pi_\epsilon(\theta', z'|w)}{\pi_\epsilon(\theta^{(t-1)}, z^{(t-1)}|w)} \frac{K(\theta^{(t-1)}|\theta')f(z^{(t-1)}|\theta^{(t-1)})}{K(\theta'|\theta^{(t-1)})f(z'|\theta')} \tag{5.7}$$

Invullen van (5.6) in (5.7) geeft

$$\frac{\pi_\epsilon(\theta', z'|w)}{\pi_\epsilon(\theta^{(t-1)}, z^{(t-1)}|w)} \frac{K(\theta^{(t-1)}|\theta')f(z^{(t-1)}|\theta^{(t-1)})}{K(\theta'|\theta^{(t-1)})f(z'|\theta')} =$$
$$\frac{\pi(\theta')f(z'|\theta')I_{A_{\epsilon,w}}(z')}{\pi(\theta^{(t-1)})f(z^{(t-1)}|\theta^{(t-1)})I_{A_{\epsilon,w}}(z^{(t-1)})} \frac{K(\theta^{(t-1)}|\theta')f(z^{(t-1)}|\theta^{(t-1)})}{K(\theta'|\theta^{(t-1)})f(z'|\theta')} =$$
$$\frac{\pi(\theta')K(\theta^{(t-1)}|\theta')}{\pi(\theta^{(t-1)})K(\theta'|\theta^{(t-1)})} I_{A_{\epsilon,w}}(z')$$

De acceptatiekans is dus min $\left(\frac{\pi(\theta')K(\theta^{(t-1)}|\theta')}{\pi(\theta^{(t-1)})K(\theta'|\theta^{(t-1)})} I_{A_{\epsilon,w}}(z'), 1\right)$. Voor het berekenen van de acceptatiekans die gebruikt wordt in het algoritme is berekening van de likelihoodfunctie niet nodig, deze valt weg, dus voldoet het aan de ABC eisen.

Het MCMC ABC-algoritme ziet er als volgt uit:

1. Kies een samenvattingsvariabele μ, een tolerantiegrens ϵ en een maat om te bepalen of de datasets genoeg overeenkomen ρ.
2. Bereken $\tilde{\mu}$ voor de geobserveerde data.
3. Gebruik het algoritme uit §4.1 om een realisatie (θ^0, z^0) uit de benadering van de a postriori verdeling te krijgen.
4. Herhaal voor $t = 1, .., Q$:
   (a) Genereer θ' uit $K(\cdot|\theta^{(t-1)})$.
   (b) Genereer z' met $f(\cdot|\theta')$ (model).
   (c) Bereken μ_t voor deze z'.
   (d) Genereer u uit een uniforme verdeling op $[0, 1]$.
   (e) Als $u \leq \frac{\pi(\theta')K(\theta^{(t-1)}|\theta')}{\pi(\theta^{(t-1)})K(\theta'|\theta^{(t-1)})}$ en $\rho(\tilde{\mu}, \mu_t) \leq \epsilon$
   neem $(\theta^{(t)}, z^{(t)}) = (\theta', z')$.
   (f) Anders
   neem $(\theta^{(t)}, z^{(t)}) = (\theta^{(t-1)}, z^{(t-1)})$.

Aangezien een Markovketen zijn begintoestand vergeet kan stap 3 overgeslagen worden, neem (θ^0, z^0) willekeurig. In dat geval moeten er wel meer simulaties uitvoerd worden voordat de keten convergeert naar de a postriori verdeling.

5.3 Eenvoudige illustratie van het MCMC ABC-algoritme

We bekijken weer het model waarin de data gegenereerd wordt door

$$X_t = \beta X_{t-1} + z_t, \text{ met } X_0 = 0, \ z_t \sim N(0, \sigma_1^2), \ t = 0, .., M$$

We hebben geobserveerde data die gegeneert is met dit model en $\beta_0 = 0.8$. Vervolgens schatten we β_0 met de MCMC ABC-methode. We voeren Q simulaties uit en als $K(\cdot|\beta^{(t-1)})$ wordt een $N(\beta^{(t-1)}, \sigma_2^2)$ verdeling genomen. Deze is symmetrisch:

$$K(\beta'|\beta^{(t-1)}) = \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp^{-\frac{1}{2\sigma_2^2}(\beta' - \beta^{(t-1)})^2} = K(\beta^{(t-1)}|\beta')$$

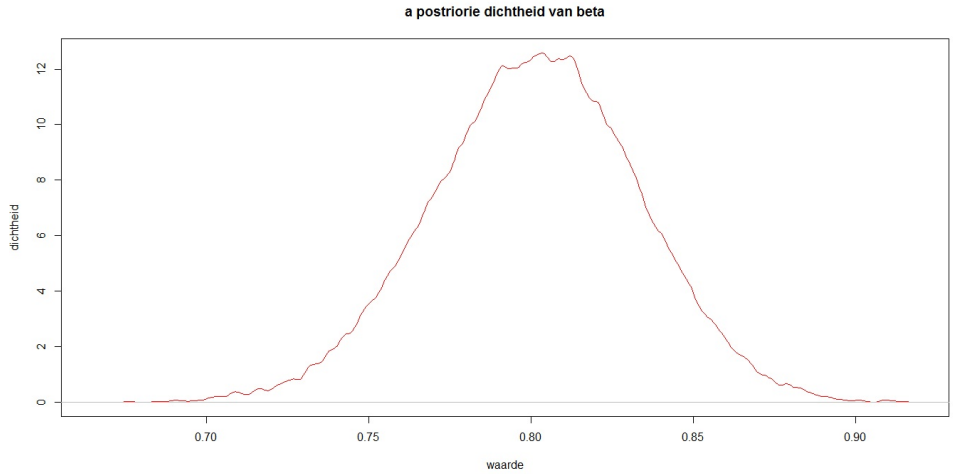
Dus wordt de acceptatiekans min $\left(\frac{\pi(\beta')}{\pi(\beta^{(t-1)})} I_{A_{\epsilon,w}}(z'), 1\right)$ en omdat zowel $\pi(\beta')$ als $\pi(\beta^{(t-1)})$ uniform op $[-1, 1]$ worden genomen is de acceptatiekans min $\left(I_{A_{\epsilon,w}}(z'), 1\right)$. Dit betekent dat in het MCMC ABC-algoritme stap 4(d) overgeslagen kan worden en dat voor stap 4(e) alleen $\rho(\tilde{\mu}, \mu_t) \leq \epsilon$ hoeft te gelden.

5.3.1 Simulatie van het algoritme in R

We kiezen:

- $\mu = \sum_{j=2}^{6}$(autocorrelatie tijdsverschil j)
- ρ is de Euclidische afstand
- $Q = 10^7$
- $M = 500$
- $\epsilon = 0.01$
- $\sigma_1 = 0.01$
- $\sigma_2 = 0.05$ dus $K(\cdot|\beta^{(t-1)})$ is een $N(\beta^{(t-1)}, 0.05^2)$ verdeling.

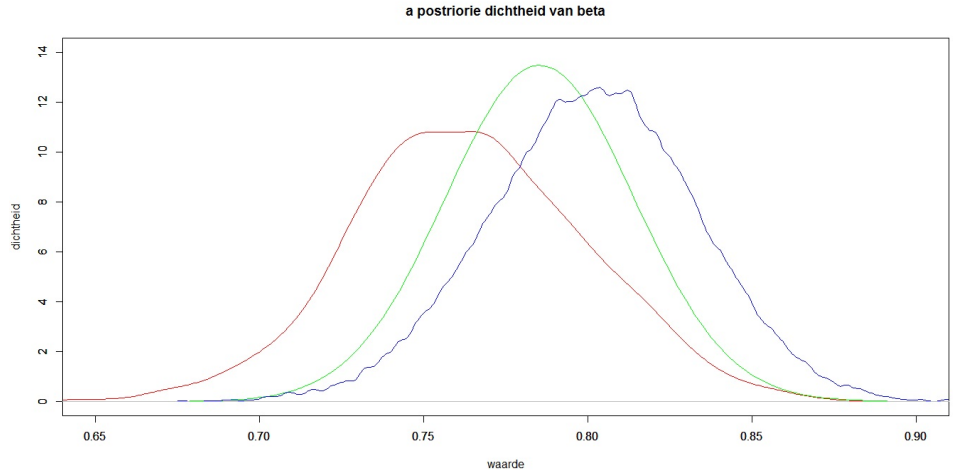
5.3.2 Resultaten



Figuur 5.1: MCMC ABC benadering van de a postriori verdeling van beta

Het algoritme met $Q = 10^7$ simulaties geeft als schatting voor $\hat{\beta}_0 = 0.801$, de rekentijd is ongeveer 8.5 uur. Zie figuur-5.1 voor de dichtheid van de benadering van de a postriori verdeling.

De pieken die je terugziet in de dichtheid worden veroorzaakt door het feit dat als een kandidaat niet voldoet ($\rho(\tilde{\mu}, \mu_t) \geq \epsilon$), de vorige waarde vastgehouden wordt. Als dit meerdere keren achter elkaar gebeurt dan krijg je een grotere dichtheid rond die waarde van β in de a postriori verdeling. Uit het feit dat de piekvormigheid sterk afneemt als Q toeneemt verwachten we dat dit effect voor Q groot genoeg helemaal weg is. Het is qua rekentijd helaas niet haalbaar om dit te verifiëren.



Figuur 5.2: MCMC ABC benadering (blauw), ABC benadering (rood) en numerieke benadering van de exacte a postriori verdeling (groen) van beta

Zie figuur-5.2 voor een vergelijking met de ABC-benadering en de numerieke benadering van de exacte a postriori verdeling (zie §4.2). Hierin is te zien dat de MCMC ABC-methode de exacte a postriori verdeling beter benaderd dan de ABC-methode . Dit wordt bevestigd door het feit dat de MCMC ABC-methode een betere schatting voor β_0 geeft ($\hat{\beta}_0 = 0.801$ vs $\hat{\beta}_0 = 0.760$). Uit de theorie verwachten we dat voor Q groot genoeg en $\epsilon \to 0$ de MCMC ABC-benadering convergeert naar de exacte a postriori verdeling.

5.4 MCMC ABC-algoritme toegepast op het model

We passen het MCMC ABC-algoritme toe op het model van Ghoelmie, Cont en Nadal (§2.1). We genereren logreturns r met een bepaalde s_0, alle andere parameters houden we contstant, dit is de geobserveerde data. Vervolgens schatten we met de MCMC ABC-methode met welke s deze geobserveerde data gegenereerd is.

Net als in §5.3 voeren we Q simulaties uit en nemen we een $N(s^{(t-1)}, \sigma^2)$ verdeling als $K(\cdot|s^{(t-1)})$. Omdat deze symmetrisch is en omdat we $\pi(s') = \pi(s^{(t-1)})$ beide uniform op $[0, 1]$ nemen is de acceptatiekans ook hier $\min\big(I_{A_{\epsilon,w}}(z'), 1\big)$. Dus we slaan in het MCMC ABC-algoritme stap 4(d) weer over en voor stap 4(e) hoeft alleen $\rho(\tilde{\mu}, \mu_t) \leq \epsilon$ te gelden.

Om convergentie naar de a postrior verdeling en een goede eerste schatting s^0 te krijgen (waardoor die convergentie ook eerder optreedt) willen we ϵ zo klein mogelijk kiezen. We worden hier bij de simulatie van het algoritme erg in beperkt door het feit dat de rekentijd snel toeneemt als ϵ kleiner wordt.

Als de beginschatting s^0 goed is, is een zo klein mogelijke σ gunstig. Is deze minder goed dan zorgt een kleine σ ervoor dat het langer duurt voordat er convergentie optreedt. We kiezen een σ die proefondervindelijk goede resultaten oplevert voor verschillende s^0.

5.4.1 Simulatie van het algoritme in R

We kiezen

- $s_0 = 0.4$
- Parameters voor het model (zie §2.1): $T = 1000$, $N = 500$, $D = 0.001$, $\lambda = 10$, $g(z) = \frac{z}{\lambda}$ en $\theta_i(0) = \theta_0 = \frac{1}{2}$.
- $\rho(\tilde{\mu}, \mu_t) = \sum_{i=2}^{10} |\text{verschil in autocorrelatie } r^2 \text{ op tijdsverschil i}|$
- $Q = 10000/20000$
- $\epsilon = 0.3$
- $\sigma = 0.1$ dus $K(\cdot|s^{(t-1)})$ is een $N(s^{(t-1)}, 0.1^2)$ verdeling.

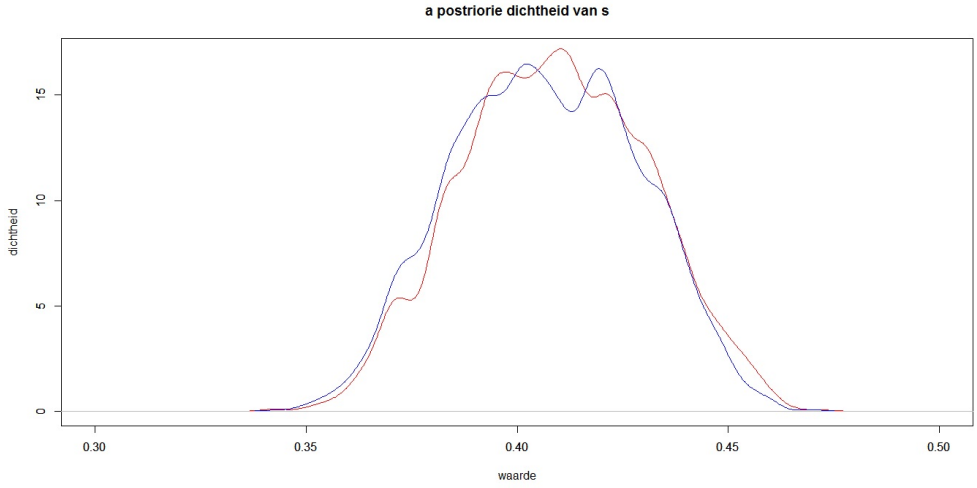
5.4.2 Resultaten

Het algoritme met $Q = 10000$ simulaties geeft als schatting voor $\hat{s_0} = 0.409$, de rekentijd bedraagt bijna 12 uur. $Q = 20000$ simulaties geeft $\hat{s_0} = 0.407$, de rekentijd neemt hier proportioneel toe en bedraagt ongeveer 24 uur.

Q	$\hat{\mathbf{s_0}}$	**rekentijd (uur)**
10000	0.409	12
20000	0.407	24

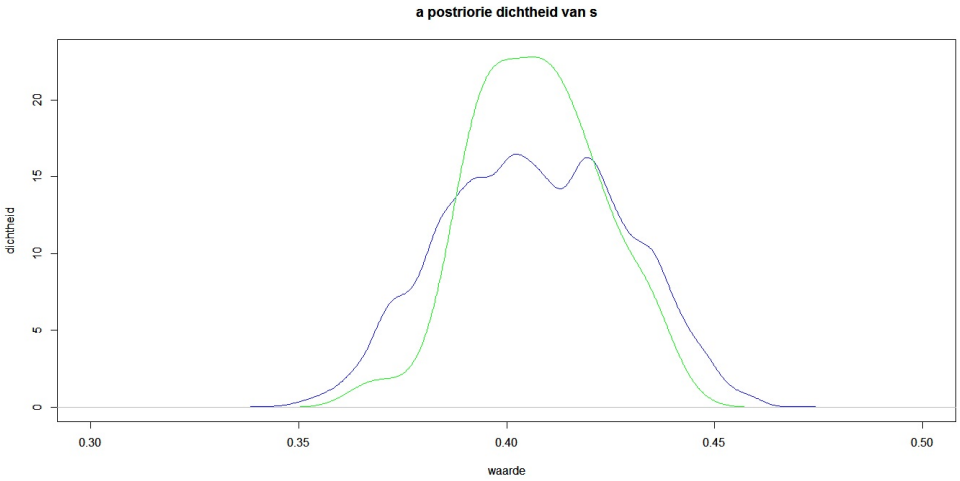
Tabel 5.1: resultaten MCMC ABC methode op model

Zie figuur-5.3 voor de benadering van de a postriori dichtheid van s voor $Q = 10000$ en $Q = 20000$. Er is bij deze MCMC ABC benaderingen nog een sterkere piekvormigheid te zien dan bij de resultaten van het eenvoudige model (§5.3.2, figuur-5.1). Dit is komt door het feit dat ϵ groter is gekozen, dit om de rekentijd te beperken. Hierdoor wordt een kandidaat voor s minder snel geaccepteerd waardoor er vaker dezelfde waarde achter elkaar voorkomt in de tijdreeks, dit zorgt voor piek in de dichtheid rond die waarde.



Figuur 5.3: MCMC ABC benaderingen van de a postriori verdeling van s (Q=10000, rood en Q=20000, blauw)

De MCMC ABC schattingen voor s_0 zijn niet beter dan schattingen met het ABC-algoritme, deze had als beste schatting ook $\hat{s_0} = 0.407$. De benadering van de a postriori verdeling ligt voor de ABC methode ook dichter rond de gekozen waarde voor $s_0 = 0.4$ dan voor de benadering met de MCMC methode (Q=20000), zie figuur-5.4. Dit wordt veroorzaakt doordat ϵ te groot en Q te klein is gekozen waardoor het algoritme nog niet genoeg naar de a postriori verdeling convergeert. Het is helaas door de lange rekentijd niet mogelijk om dit te verbeteren.



Figuur 5.4: MCMC ABC (blauw) en ABC (groen) benadering van de a postriori verdeling van s

Hoofdstuk 6

Tweede uitbreiding op het ABC-algoritme

In de voorgaande twee methoden bleek het kiezen van een zo'n klein mogelijke ϵ al snel te zorgen voor lange rekentijden. In het begin van het algoritme ϵ relatief groot kiezen zodat je snel de slechtste kandidaten wegfiltert en dan steeds ϵ verkleinen op het moment dat je betere kandidaten overhoudt, zou de rekentijd moeten verbeteren. Dit idee wordt uitgewerkt in de volgende ABC-methode gebasseerd op sequential Monte Carlo (SMC ABC) [10].

6.1 SMC ABC-algoritme

Bij het SMC ABC-algoritme wordt de a postriori verdeling stapsgewijs benaderd door middel van 'tussenverdelingen', deze worden populaties genoemd. Er zijn $p = 1, .., P$ populaties met elk een ϵ_p, er geldt $\epsilon_1 > \epsilon_2 > .. > \epsilon_P$. We trekken uit de a priori verdeling $\pi(\theta)$ totdat er N kandidaten zijn geaccepteerd (volgens het ABC-algoritme met ϵ_1). We berekenen voor alle geaccepteerde kandidaten een gewicht [11], dit is populatie 1. Vervolgens trekken we θ^* uit populatie 1 en genereren we de nieuwe kandidaat $\theta^{**} \sim K(\theta|\theta^*)$ met K een transitiekern tot er N keer een θ^{**} geaccepteerd is (volgens het ABC-algoritme met ϵ_2). We berekenen voor elke geaccepteerde kandidaat een gewicht, dit is populatie 2. We doen dit todat er N kandidaten zijn geaccepteerd in de laatste populatie P, deze populatie is de benadering voor de a postriori verdeling.

We duiden de i-de geaccepteerde kandidaat in populatie p aan met $\theta_p^{(i)}$. Het SMC ABC-algoritme ziet er als volgt uit:

1. Kies een samenvattingsvariabele μ, de tolerantiegrenzen $\epsilon_1 > \epsilon_2 > .. > \epsilon_P$ en een maat om te bepalen of de datasets genoeg overeenkomen ρ.

2. Bereken $\tilde{\mu}$ voor de geobserveerde data.

3. Herhaal voor $p = 0, .., P$

   - Herhaal voor $i = 1, .., N$
     (a) Als $p = 0$: trek θ^{**} uit $\pi(\theta)$. Anders, trek θ^* uit de vorige populatie $\{\theta_{p-1}^{(1)}, .., \theta_{p-1}^{(N)}\}$ met gewichten $\{w_{p-1}^{(1)}, .., w_{p-1}^{(N)}\}$ en genereer θ^{**} uit $K(\theta|\theta^*)$.
     (b) Als $\pi(\theta^{**}) = 0$: herhaal (a).
     (c) Genereer data z' met $f(\cdot|\theta^{**})$ (model).
     (d) Bereken μ^{**} voor deze z'.
     (e) Als $\rho(\tilde{\mu}, \mu^{**}) \geq \epsilon_p$: herhaal (a),(b),(c) en (d).
     (f) Zet $\theta_p^{(i)} = \theta^{**}$ en bereken het gewicht voor $\theta_p^{(i)}$:
     $$w_p^{(i)} = \begin{cases} 1, & \text{als } p = 0 \\ \frac{\pi(\theta_p^{(i)})}{\sum_{j=1}^{N} w_{p-1}^{(j)} K(\theta_p^{(i)}|\theta_{p-1}^{(j)})}, & \text{als } p > 0 \end{cases}$$
   - Normalizeer de gewichten $w_p^{(1)}, .., w_p^{(N)}$.

De gewichten in dit algoritme komen voort uit de theorie van important sampling. Hierbij is het doel om trekkingen te genereren uit de kansdichheid van een stochastische variabele X, deze noemen we de target distributie $\tau(x)$. Stel dat het lastig is om te trekken uit $\tau(x)$. Daarom introduceren we g, dit is de kansdichtheid van een stochastische variabele, de proposal X^{prop}. Hierbij is het trekken uit X^{prop} wel eenvoudig.

De verwachting van een functie $f : \mathbb{R} \to \mathbb{R}$ wordt gegeven door

$$E[f(x)] = \int f(x)\tau(x)dx = \int f(x) \underbrace{\frac{\tau(x)}{g(x)}}_{w(x)} g(x)dx = \int f(x)w(x)g(x)dx$$

$$= E[f(X^{\text{prop}})w(X^{prop})] \approx \frac{1}{N}\sum_{i=1}^{N} f(X_i^{\text{prop}})w(X_i^{\text{prop}})$$

Dus we benaderen de verdeling van X met een discrete verdeling $(X_1^{\text{prop}}, .., X_N^{\text{prop}})$ met gewichten $(w_1^{\text{prop}}, .., w_N^{\text{prop}})$. We normaliseren de gewichten,

$$\tilde{w_i} = \frac{w(X_i^{\text{prop}})}{\sum_{j=1}^{N} w(X_j^{\text{prop}})}$$

Dan wordt de important sampling schatter voor $E[f(x)]$ gegeven door $\sum_{i=1}^{N} \tilde{w_i} f(X_i^{\text{prop}})$. Dit is met name nuttig omdat f vaak alleen bekend is op een multiplicatieve contstante na.

In ons geval hebben we als target distributie τ de a postriori verdeling $\pi_\epsilon(\theta, z|w)$, met w de geobserveerde data en z de gegenereerde data. We weten $\pi_\epsilon(\theta, z|w) \propto f(z|\theta)\pi(\theta)I_{A_{\epsilon,w}}(z)$, zie vergelijking (5.6). Hierin is $A_{\epsilon,w}$ de verzameling van alle z gegenereert met een θ die voor deze gekozen ϵ en deze w geaccepteerd wordt. $f(z|\theta)$ is het genereren van data z met het model met θ als parameter en $\pi(\theta)$

is de a priori verdeling.

De proposal g is $\sum_{i=1}^{N} \tilde{w_{i-1}} K(\theta'|\theta^{(i-1)}) f(z'|\theta')$, met θ' de kandidaat voor θ en z' de gegenereerde data hiermee. Het gewicht voor een kandidaat θ' wordt dus

$$\frac{\tau}{g} = \frac{\pi_\epsilon(\theta', z'|w)}{\sum_{i=1}^{N} \tilde{w_{i-1}} K(\theta'|\theta^{(i-1)}) f(z'|\theta')} = \frac{f(z'|\theta')\pi(\theta') I_{A_{\epsilon,w}}(z')}{f(z'|\theta') \sum_{i=1}^{N} \tilde{w_{i-1}} K(\theta'|\theta^{(i-1)})} = \frac{\pi(\theta')}{\sum_{i=1}^{N} \tilde{w_{i-1}} K(\theta'|\theta^{(i-1)})}$$

In deze laatste stap is $I_{A_{\epsilon,w}}(z') = 1$ omdat in het algoritme θ' al een geaccepteerde kandidaat is, dus z' zit in $A_{\epsilon,w}$. Bij het berekenen van de gewichten in het algoritme valt de likelihoodfunctie weg, dus voldoet het aan de ABC eisen.

6.2 Eenvoudige illustratie van het SMC ABC-algoritme

We bekijken opnieuw het model waarin de data gegenereerd wordt door

$$X_t = \beta X_{t-1} + z_t, \text{ met } X_0 = 0, \ z_t \sim N(0, \sigma_1^2), \ t = 0, .., M$$

We hebben geobserveerde data die gegeneert is met dit model en $\beta_0 = 0.8$. Vervolgens schatten we β_0 met de SMC ABC-methode. Er worden $p = 0, .., P$ populaties genomen met in elke populatie $i = 1, .., N$ geaccepteerde kandidaten.

Als transitiekern $K(\beta|\beta^*)$ nemen we een $N(\beta^*, \sigma_2^2)$ verdeling. De a priori verdeling waar we β^* uit trekken is een uniforme verdeling op $[-1, 1]$, dus $\pi(\beta) = \frac{1}{2} \ \forall \beta \in [-1, 1]$. We berekenen het gewicht voor $p > 0$ dus als volgt:

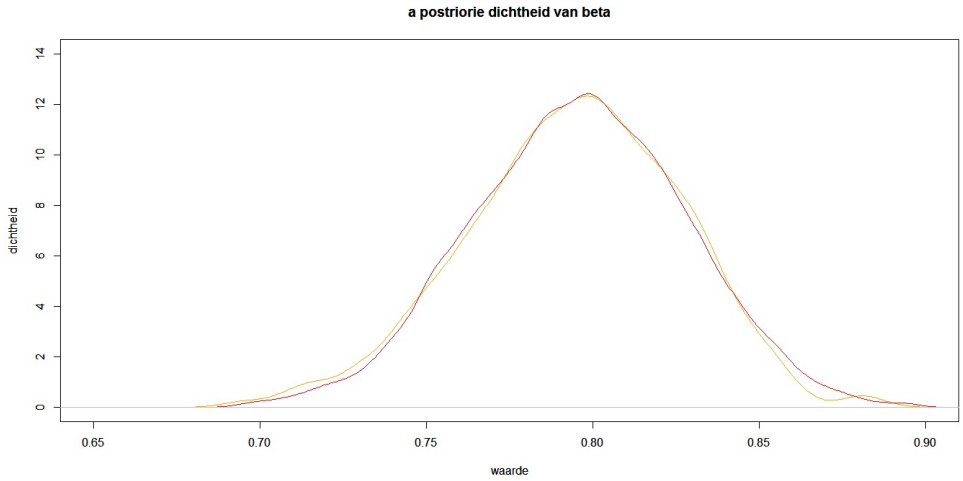
$$\frac{\pi(\beta_p^{(i)})}{\sum_{j=1}^{N} w_{p-1}^{(j)} K(\beta_p^{(i)}|\beta_{p-1}^{(j)})} = \frac{\frac{1}{2} 1_{[-1,1]}(\beta_p^i)}{\sum_{j=1}^{N} w_{p-1}^{(j)} g(\beta_p^{(i)})} \text{ met } g(x) = \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp^{-\frac{1}{2\sigma_2^2}(x-\beta_{p-1}^{(j)})^2}$$

De tolerantiegrenzen $\epsilon_1, .., \epsilon_P$ worden zo gekozen dat $\epsilon_p = 0.7\epsilon_{p-1}$ met een bepaalde ϵ_0.

6.2.1 Simulatie van het algoritme in R

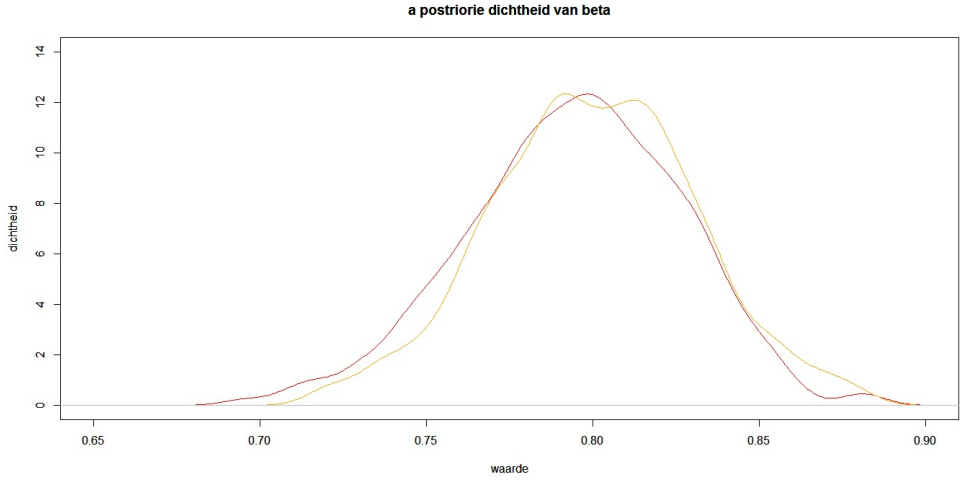
- $M = 500$
- $\sigma_1 = 0.01$
- $\mu = \sum_{j=2}^{6}$(autocorrelatie tijdsverschil j)
- ρ is de Euclidische afstand
- $N = 1000/10000$
- $P = 10/12$
- $\epsilon_0 = 0.1$
- $\sigma_2 = 0.05$ dus $K(\cdot|\beta^*)$ is een $N(\beta^*, 0.05^2)$ verdeling.

6.2.2 Resultaten



Figuur 6.1: SMC ABC benaderingen van de a postriori verdeling van beta (P=10, N=1000 (oranje)/ N=10000 (rood))

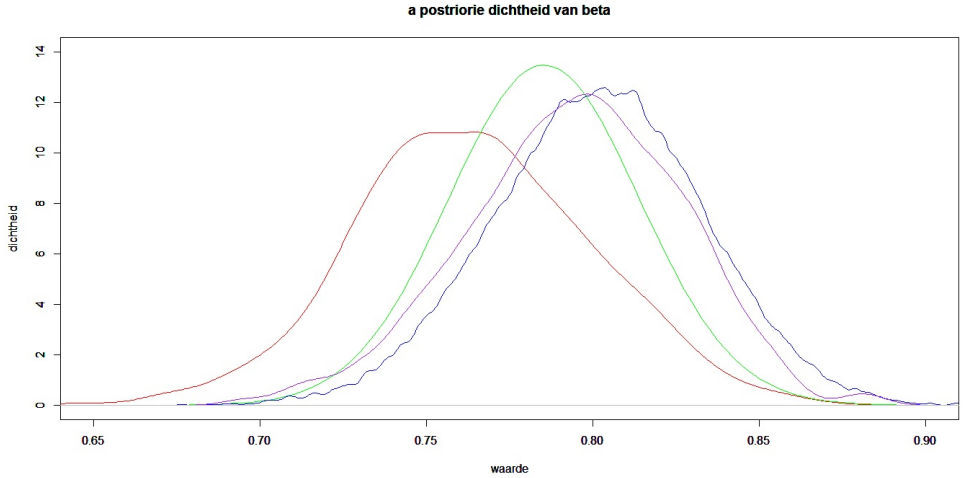
Zie figuur-6.1 voor de benadering van de a postriori verdeling van beta met $P = 10$ en $N = 1000/N = 10000$, de rekentijd was 24 minuten en respectievelijk 362 minuten. Omdat het verschil in de benadering van de a postriori verdeling klein is en het verschil in rekentijd groot, kiezen we om verdere resultaten te genereren met N=1000.



Figuur 6.2: SMC ABC benaderingen van de a postriori verdeling van beta (P=10 (rood)/ P=12 (oranje), N=1000)

Een simulatie van het algoritme met $P = 10$ en dus $\epsilon_P = 0.0040$ geeft als schatting $\hat{\beta}_0 = 0.794$, een simulatie met $P = 12$ en $\epsilon_P = 0.0019$ geeft $\hat{\beta}_0 = 0.801$ en heeft ongeveer 45 minuten rekentijd, zie figuur-6.2.

In figuur-6.3 zien we de SMC ABC benadering (met P=10, N=1000), de MCMC ABC benadering, de ABC benadering en de numerieke benadering van de exacte a postriori verdeling van beta. Hieruit volgt dat de SMC en MCMC ABC benaderingen ongeveer even dicht bij de exacte a postriori verdeling liggen. Het voordeel van de SMC ABC methode ten opzichte van de MCMC ABC methode is dat hij sneller is, de rekentijd was 24 minuten tegenover 8.5 uur.



Figuur 6.3: SMC ABC benadering (paars), MCMC ABC benadering (blauw), ABC benadering (rood) en numerieke benadering van de exacte a postriori verdeling (groen) van beta

6.3 SMC ABC-algoritme toegepast op het model

We passen het SMC ABC-algoritme toe om te schatten met welke s_0 een geobserveerde dataset is gegenereerd door het model van Ghoelmie, Cont en Nadal (§2.1), de rest van de parameters zijn constant genomen.

Er worden weer $p = 0, .., P$ populaties genomen en in elke populatie zitten $i = 1, .., N$ geaccepteerde kandidaten. De tolerantiegrenzen zijn $\epsilon_p = 0.9\epsilon_{p-1}$ met een bepaalde ϵ_0. De a priori verdeling voor s is de uniforme verdeling op $[0, 1]$, hieruit trekken we s^*. Vervolgens genereren we s^{**} uit de transitiekern $K(s|s^*)$, hiervoor nemen we een $N(s^*, \sigma^2)$ verdeling. Het gewicht van een geaccepteerde

kandidaat berekenen we nu als volgt:

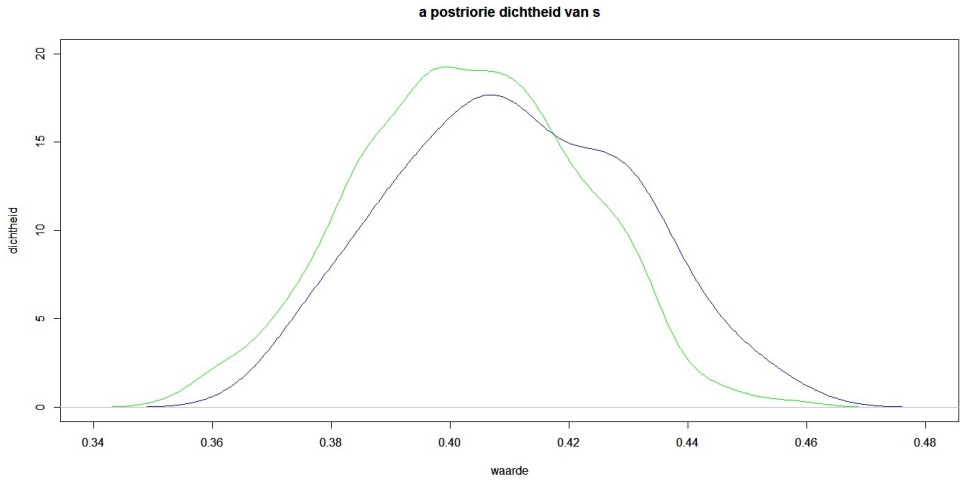
$$\frac{\pi(s_p^{(i)})}{\sum_{j=1}^{N} w_{p-1}^{(j)} K(s_p^{(i)}|s_{p-1}^{(j)})} = \frac{1_{[0,1]}(s_p^{(i)})}{\sum_{j=1}^{N} w_{p-1}^{(j)} g(s_p^{(i)})} \text{ met } g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2\sigma^2}(x - s_{p-1}^{(j)})^2}$$

6.3.1 Simulatie van het algoritme in R

We kiezen

- $s_0 = 0.4$
- Parameters voor het model (zie §2.1): $T = 1000$, $N = 500$, $D = 0.001$, $\lambda = 10$, $g(z) = \frac{z}{\lambda}$ en $\theta_i(0) = \theta_0 = \frac{1}{2}$.
- $\rho(\tilde{\mu}, \mu_t) = \sum_{i=2}^{10} |\text{verschil in autocorrelatie } r^2 \text{ op tijdsverschil i}|$
- $N = 100/800$
- $P = 7/10$
- $\epsilon_0 = 0.45$
- $\sigma = 0.01$ dus $K(\cdot|s^*)$ is een $N(s^*, 0.01^2)$ verdeling.

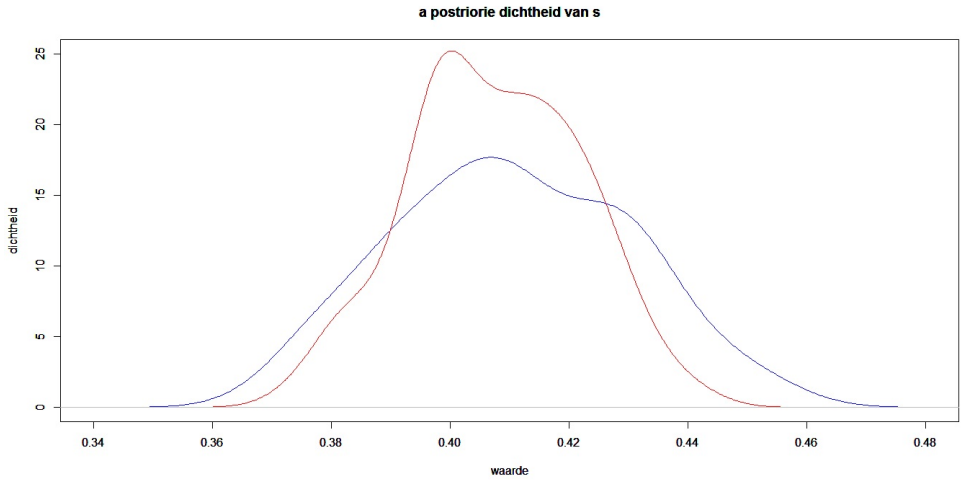
6.3.2 Resultaten



Figuur 6.4: SMC benadering van de a postriori verdeling van s, P=7, N=100 (blauw)/ N=800 (groen)

Bij de keuze van $P = 7$ populaties met in elke populatie $N = 100$ geaccepteerde kandidaten voor s geeft het algoritme als schatting $\hat{s_0} = 0.410$. De rekentijd voor deze schatting bedraagt 4 uur en 40 minuten. Nemen we nu $N = 800$ geaccepteerde kandidaten per populatie dan is de schatting $\hat{s_0} = 0.403$. De rekentijd

is lang, ongeveer 31 uur en 40 minuten. Zie figuur-6.4 voor de benadering van de a postriori verdeling. Het verschil in de twee verdelingen is relatief klein in vergelijking met het verschil in rekentijd. Omdat we deze laatste wilde beperken bekijken we de verdeling van een simulatie met $N = 100$.



Figuur 6.5: SMC benadering van de a postriori verdeling van s, N=100, P=7 (blauw)/ P=10 (rood)

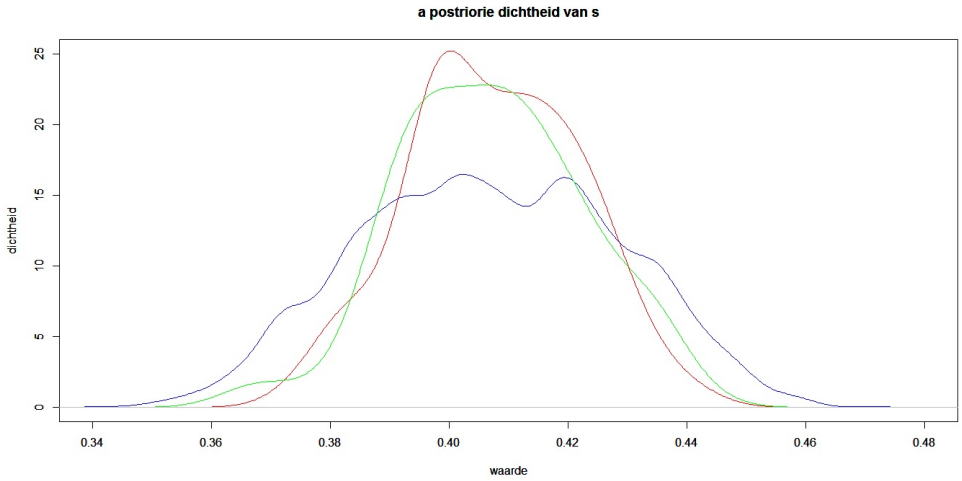
In figuur-6.5 zien we een benadering van de a postriori verdeling van s met het SMC ABC-algoritme met $P = 7$ en $P = 10$. De rekentijd van de simulatie met $P = 10$ bedraagt ongeveer de 12 uur en de schatting is $\hat{s_0} = 0.408$.

N	**P**	$\hat{\mathbf{s_0}}$	**rekentijd (uur)**
100	7	0.410	4.67
800	7	0.404	31.67
100	10	0.408	12

Tabel 6.1: resultaten SMC ABC methode op model

Het SMC algoritme met $N = 100$ geaccepteerde kandidaten in elke van de $P = 10$ populaties geeft een benadering van de a postriori verdeling die nog iets beter rond s_0 ligt dan de ABC benadering, zie figuur-6.6.

De rekentijd is met 12 uur korter dan de ongeveer 16 uur die een simulatie met het ABC-algoritme duurde. In vergelijking met de MCMC benadering is de SMC benadering duidelijk beter en de rekentijd is met de helft verkort, van 24 naar 12 uur. Dit terwijl de $\epsilon_{10} = 0.17$ die gebruikt is om de laatste populatie te genereren bij de SMC benadering een stuk kleiner is dan de $\epsilon = 0.3$ die gebruikt is bij de MCMC ABC benadering. De methode werkt dus om de rekentijd te verkorten en zo ϵ kleiner te kunnen kiezen.

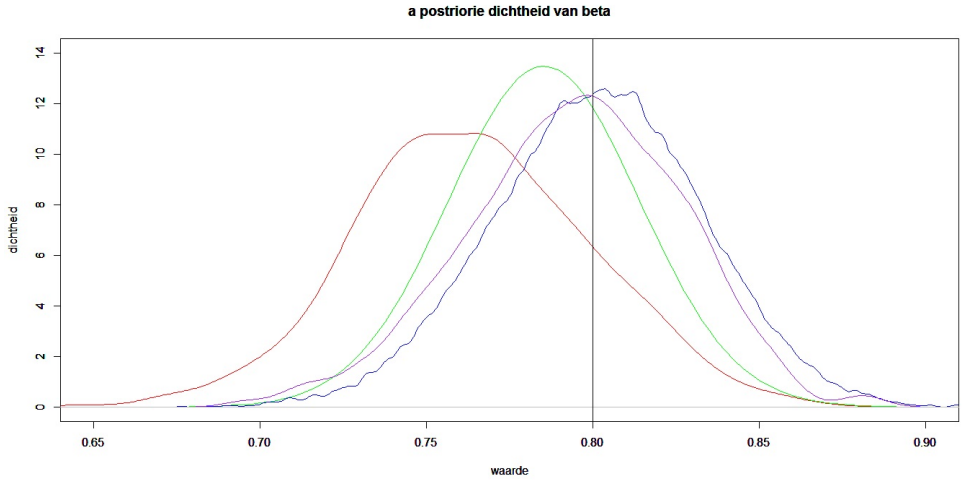


Figuur 6.6: SMC ABC benadering (rood), MCMC ABC benadering (blauw), ABC benadering (groen) van de a postriori verdeling van s

Hoofdstuk 7

Conclusie

In hoofdstuk 2 hebben we gezien dat er 3 veel voorkomende eigenschappen zijn in de logreturns van aandelenkoersen. De eigenschap dat aandelenkoersen volatiliteitsclusters (§3.2.1) vertonen komt niet terug in de data die gegeneert is met het model uit hoofdstuk 1. Wel vertonen deze data dikke staarten zoals beschreven in §3.2.2, maar in mindere mate dan echte aandelenkoerzen. Ook is de data net als echte koerzen niet lineair afhankelijk in de tijd, zie §3.2.3.



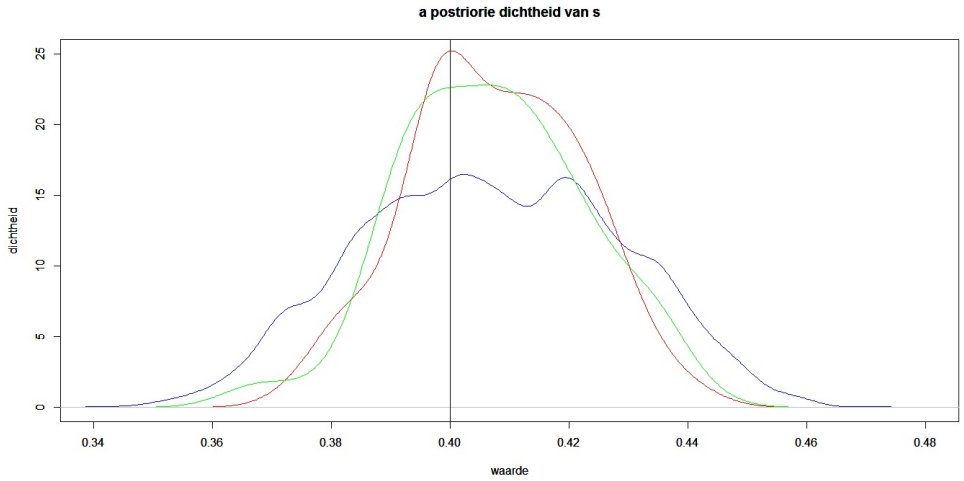
Figuur 7.1: ABC (rood), MCMC ABC (blauw), SMC (paars) en numerieke benadering van de exacte a postriori verdeling (groen) van beta

	schatting	**rekentijd (min)**
ABC	0.760	89
MCMC ABC	0.801	509
SMC ABC	0.794	45

Tabel 7.1: schattingen voor $\beta_0 = 0.8$

Om voor een eenvoudig model de parameter $\beta_0 = 0.8$ waarmee geobserveerde data is gegenereert te schatten, kunnen we het beste de SMC ABC-methode ge-

bruiken. Deze geeft een benadering van de a postriori verdeling die het dichtst bij een numerieke benadering van de exacte a postriori verdeling ligt. De schatting $\hat{\beta}_0$ met deze methode is iets minder goed dan die met de MCMC ABC-methode maar dit verschil is zo klein dat dit ook aan toevalsfactoren kan liggen en de rekentijd is veel sneller, zie figuur-7.1 en tabel-7.1.



Figuur 7.2: ABC (groen), MCMC ABC (blauw), SMC (rood) benadering van de a postriori verdeling van s

	schatting	**rekentijd (min)**
ABC	0.4071	948
MCMC ABC	0.4067	1445
SMC ABC	0.4075	730

Tabel 7.2: schattingen voor $s_0 = 0.4$

Als we de parameter $s_0 = 0.4$ van het model uit hoofdstuk 1 willen schatten aan de hand van een geobserveerde dataset, is de exacte a postriori verdeling onbekend. Wel weten we dat die rond de 0.4 moet liggen. De schattingen $\hat{s_0}$ verschillen voor dit probleem erg weinig, maar aan de hand van de rekentijd kiezen we ook hier voor de SMC ABC-methode, zie figuur-7.2 en tabel-7.2.

Hoofdstuk 8

Bijlage

8.1 Code §2.2

```
t<-10000 #aantal tijdsstappen
N<-1500#aantal agents
s<-0.1 # 1/gemiddelde update periode van threshold
D<-0.001 #standaard deviatie inkomende informatie stroom
lambda<-10 #marktdiepte

#u(t) iid random variables uniform op [0,1], voor updaten threshold
u<-matrix(runif(t*N,0,1),t,N)
#inkomende informatiestroom
e<-matrix(rnorm(t,0,D),t,1)

#beginverdeling voor de thresholds, allemaal 0.5 genomen
#let op dit is een van de parameters
theta0<- matrix(0.5,1,N)

theta<-matrix(0,t,N) #lege matrix voor de thresholds
phi<-matrix(0,t,N) #lege matrix voor de vraag
Z<-matrix(0,t,1) #lege vector voor excess demands
r<-matrix(0,t,1) #lege vector voor de returns

theta[1,]=theta0 #eerste rij van theta is beginverdeling thresholds

#eerste tijdstap
for(i in 1:N){if(e[1,1]< -1*theta[1,i])phi[1,i]=-1
              if(e[1,1]>theta[1,i])phi[1,i]=1} #eerste rij van phi maken
Z[1,1]=sum(phi[1,]) #eerste waarde Z
)#eerste waarde return, hoe je dit berekend is ook een parameter
r[1,1]=Z[1,1]/(N*lambda

#rest model
for(i in 2:t){
  for(j in 1:N){
```

```
    if(u[i,j]<s)theta[i,j]=abs(r[i-1,1])
    if(u[i,j]>=s)theta[i,j]=theta[i-1,j] #theta verder maken
    if(e[i,1]< -1*theta[i,j])phi[i,j]=-1 #phi daaruit maken
    if(e[i,1]> theta[i,j])phi[i,j]=1
  }
  Z[i,1]=sum(phi[i,])
  r[i,1]=Z[i,1]/(N*lambda)
}

#de prijs maken:
p0<-0.80#beginprijs, is parameter
p<-matrix(0,t,1)#lege vector voor prijzen
p[1,1]=p0
for(i in 2:t){
  p[i,1]=exp(r[i,1])*p[i-1,1]
}
plot(p,type="l",main="prijs, p0=0.8",xlab="t",ylab="prijs")

#returns plotten
par(mfrow=c(1,1))
plot(r,type = "s",main='logreturn',xlab="t")

#dichtheid van r plotten:
d<-density(r)
plot(d,main="dichtheid van het rendement")

#marktactiviteit
phi2<-matrix(0,t,N) #phi2 maken waar zowel kopen als verkopen waarde 1 heeft
for(i in 1:t){
  for(j in 1:N){
    phi2[i,j]=abs(phi[i,j])
  }
}
#vector met aantal agents die een handeling verrichten op tijdstip t
m<-numeric(t)
for(i in 1:t){
  m[i,1]=sum(phi2[i,])
}
plot(m,type="s",main="marktactiviteit",xlab="t",ylab="m")
```

8.2 Code hoofdstuk 3

```
#prijzen erin stoppen
price1 <- get.hist.quote(instrument='PHG',start='2000-01-01',
                         quote='AdjClose') #ahold
price2 <- get.hist.quote(instrument='SNE',start='2000-01-01',
                         quote='AdjClose') #sony
```

```
price3 <-p #deze vector is gemaakt met het model in hfst 1

#plotten van de prijzen
par(mfrow=c(3,1))
plot(price1,main='prijs Philips',xlab="",ylab="")
plot(price2,main='prijs Sony',xlab="",ylab="")
plot(head(price3,length(price1)),type="l",main="gemodelleerde prijs, p0=0.8",
                                                              xlab="t",ylab="prijs")

#logreturns maken
logreturn1 <- diff(log(price1))
logreturn2 <- diff(log(price2))
logreturn3 <- diff(log(price3))
logreturn3 <-head(logreturn3,length(logreturn1))

#logreturns plotten
par(mfrow=c(3,1))
plot(logreturn1,type = "s",main='logreturn1, philips',xlab="",ylab="")
plot(logreturn2,type = "s",main='logreturn2, sony',xlab="",ylab="")
plot(logreturn3,type = "s",main='logreturn3, model',xlab="",ylab="")

#er een time series object van maken
logreturn1 <- as.ts(coredata(logreturn1))
logreturn2 <- as.ts(coredata(logreturn2))
logreturn3 <- as.ts(coredata(logreturn3))

#strip date information from returns
logreturn1=coredata(logreturn1)
logreturn2=coredata(logreturn2)
logreturn3=coredata(logreturn3)

#wat statistieken
library(moments)
mean(logreturn1) #gemiddelde
mean(logreturn2)
mean(logreturn3)

sd(logreturn1) #standaard deviatie
sd(logreturn2)
sd(logreturn3)

min(logreturn1) #minimale logreturn
min(logreturn2)
min(logreturn3)

max(logreturn1) #maximale logreturn
max(logreturn2)
max(logreturn3)

skewness(logreturn1) #scheefheid
```

```
skewness(logreturn2)
skewness(logreturn3)

kurtosis(logreturn1) #platheid
kurtosis(logreturn2)
kurtosis(logreturn3)

a<-acf(logreturn1,1) #autocorrelatie logreturn
a
a<-acf(logreturn2,1)
a
a<-acf(logreturn3,1)
a

a<-acf(logreturn1^2,1) #autocorrelatie logreturn^2
a
a<-acf(logreturn2^2,1)
a
a<-acf(logreturn3^2,1)
a

# jarque bera test op normaalheid (p-waarde)
jarque.bera.test(logreturn1)
jarque.bera.test(logreturn2)
jarque.bera.test(logreturn3)

#test voor autocorrelatie
Box.test(logreturn1, lag=20, type=c("Ljung-Box"))
Box.test(logreturn2, lag=20, type=c("Ljung-Box"))
Box.test(logreturn3, lag=20, type=c("Ljung-Box"))

 #test voor autocorrelatie van logreturn^2
Box.test(logreturn1^2, lag=20, type=c("Ljung-Box"))
Box.test(logreturn2^2, lag=20, type=c("Ljung-Box"))
Box.test(logreturn3^2, lag=20, type=c("Ljung-Box"))

#autocorrelatiefuncties maken
par(mfrow=c(3,1))
Acf(logreturn1,main="autocorrelatie return Philips")
Acf(logreturn2,main="autocorrelatie return Sony")
Acf(logreturn3,main="autocorrelatie return model")

par(mfrow=c(3,1))
Acf(logreturn1^2,main="autocorrelatie return^2 Philips")
Acf(logreturn2^2,main="autocorrelatie return^2 Sony")
Acf(logreturn3^2,main="autocorrelatie return^2 model")

Box.test(logreturn1,lag=21,type="Ljung-Box")
Box.test(logreturn2,lag=21,type="Ljung-Box")
Box.test(logreturn3,lag=21,type="Ljung-Box")
```

```
#densities bekijken
plot(density(logreturn1))
plot(density(logreturn2))

#Kolmogorov-Smirnov test

ks.test(logreturn1,'pnorm',mean(logreturn1),sd(logreturn1))
ks.test(logreturn2,'pnorm',mean(logreturn2),sd(logreturn2))
ks.test(logreturn3,'pnorm',mean(logreturn3),sd(logreturn3))

#resultaten hiervan:

#data:  logreturn1
#D = 0.0647, p-value = 1.914e-12

#data:  logreturn2
#D = 0.0566, p-value = 1.271e-09

#data:  logreturn3
#D = 0.087, p-value < 2.2e-16


#QQplots maken van de logreturns tov de normale verdeling
par(mfrow=c(3,1))
qq.plot(logreturn1,main="QQplot return philips tov normale verdeling",
 xlab = "normale verdeling", ylab = "logreturn", envelope=F)
qq.plot(logreturn2,main="QQplot return sony tov normale verdeling",
 xlab = "normale verdeling", ylab = "logreturn",envelope=F)
qq.plot(logreturn3,main="QQplot return model tov normale verdeling",
 xlab = "normale verdeling", ylab = "logreturn",envelope=F)

#QQplots maken van de logreturns tov de student-t verdeling met verschillende df:
# * degrees of freedom
par(mfrow=c(3,1))
qq.plot(logreturn1,distribution="t",df=6,
main="QQplot return philips tov student-t met 6 df",envelope=F)
qq.plot(logreturn2,distribution="t",df=5,
main="QQplot return sony tov student-t met 5 df",envelope=F)
qq.plot(logreturn3,distribution="t",df=7,
main="QQplot return model tov student-t met 7 df",envelope=F)

#plaatje van momenten te maken,
# kijken of die convergeert om index van de tail te bepalen
#moment ervoor
par(mfrow=c(3,1))
plot(1:length(logreturn1),(cumsum(logreturn1^3)/(1:length(logreturn1))),
main="3de moment logreturn philips",type="l",xlab="t",ylab="")
plot(1:length(logreturn2),(cumsum(logreturn2^4)/(1:length(logreturn2))),
main="5de moment logreturn sony",type="l",xlab="t",ylab="")
```

```
plot(1:length(logreturn3),(cumsum(logreturn3^8)/(1:length(logreturn3))),
main="6de moment logreturn model",type="l",xlab="t",ylab="")

#moment erna
par(mfrow=c(3,1))
plot(1:length(logreturn1),(cumsum(logreturn1^4)/(1:length(logreturn1))),
main="4de moment logreturn philips",type="l",xlab="t",ylab="")
plot(1:length(logreturn2),(cumsum(logreturn2^5)/(1:length(logreturn2))),
main="6de moment logreturn sony",type="l",xlab="t",ylab="")
plot(1:length(logreturn3),(cumsum(logreturn3^9)/(1:length(logreturn3))),
main="7de moment logreturn model",type="l",xlab="t",ylab="")
```

8.3 Code §4.2

8.3.1 Simulatie

```
#tijd meten
ptm<-proc.time()

#parameters en vectoren enzo

M<-500   #het aantal tijdsstappen waar we informatie over hebben in de data
Q<-10^6   #het aantal simulaties we doen om beta0 te schatten
sigma<-0.01  #sd van de ruis
beta0<-0.8  #beta van de data, deze willen we benaderen met de schatting
X0<-0 #beginwaarde X

#vector met voor elke simulatie een beta uit een unif(-1,1) verdeling prior voor beta
betaY<-runif(Q,-1,1)

#'data' genereren die we gaan gebruiken:
# X(t) = 0.8X(t-1) + zet(t), X(0)=0, zet N(0,sigma^2) verdeeld

 #vector z met N(0,sigma^2) verdeelde waarden voor elke t, voor X
zetX<-rnorm(M,0,sigma)
#lege vector om de X'jes in te zetten straks
X<-numeric(M)

X[1]=X0
for(i in 2:M){
  X[i] = beta0*X[i-1] + zetX[i]
}

#summery statistic voor X berekenen
 #acf voor alle tijdsverschillen, en zorgen dat je kunt rekenen met die waarden
cX<-acf(X,plot=FALSE)$acf
#waarden optellen voor dit is je summary statistic
```

```
CX<-sum(cX[2:6])

#Nu doen alsof we beta0 niet meer weten en de verdeling hiervan willen benaderen

#lege matrix  om straks summary statistics en beta's in op te slaan
D<-matrix(0,Q,2)

for(i in 1:Q){

  #1+2 beta trekken uit uniforme verdeling en data Y genereren
  zetY<-rnorm(M,0,sigma)    #vector z met N(0,sigma^2) waarden voor elke t, voor Y

  Y<-numeric(M)             #lege vector om Y in te maken
  Y[1]=0                    #Y0=0

  for(j in 2:M){
    Y[j] = betaY[i]*Y[j-1] + zetY[j]
  }

  #3 summary statistic CY voor die Y berekenen
  cY<-acf(Y,plot=FALSE)$acf           #autocorrelaties voor Y berekenen
  CY<-sum(cY[2:6])                    #summary statistic voor Y

  #4 afstand tussen CX en CY berekenen
  d<-abs(CX-CY)

  #5 deze afstand opslaan in de vector D samen met de bijbehorende beta
  D[i,1]=d
  D[i,2]=betaY[i]

}

#kolom 1 van D in oplopende volgorde zetten,
# bijbehorende beta's (kolom 2) mee verplaatsen
D<-D[order(D[,1]),]

#vector met beta's op volgorde van degene met kleinste 'fout' in summary statistic
#tot degene met de grootste 'fout'in summary statistic
D1<-D[,2]

Opt<-head(D1,1000) # vector met 1000 beste beta's

#verdeling beste beta's, (=>a postriori verdeling)
plot(density(Opt),main="a postriorie dichtheid van beta",xlab="waarde",
 ylab="dichtheid",col="red",xlim=c(0.55,0.9),ylim=c(0,14))
par(new=TRUE)
#numerieke benadering exacte a postriori verdeling
plot(post,col="green",xlim=c(0.55,0.9),ylim=c(0,14),xlab="",ylab="")

mean(Opt)          #verwachting van beta, de 'schatting'
```

```
#totale rekentijd berekenen
proc.time()-ptm
```

8.3.2 Functie: post

```
post<-function(beta){
  #functie die numeriek a postriori verdeling van beta gegeven X_1,..,X_M berekend
  #M, sigma en X1,..,XM moeten wel in het geheugen staan!

  #teller
  tel<-teller(beta)

  #noemer
  noem<-integrate(teller,-1,1)$val

  p<-(tel/noem)
  return(p)
}
```

8.3.3 Functie: teller

```
teller<-function(beta){

  #M, sigma en X1,..,XM moeten wel in het geheugen staan!

  #som van (X(t)-beta*X(t-1))^2 berekenen
  s2<-0
  for(i in 2:M)
  {
    s2 = s2 + (X[i]-beta*X[i-1])^2
  }

  #daadwerkelijke teller berekenen
  t<-0
  if((-1<=beta)&&(beta<=1)){
    t<-exp((-1/(2*sigma^2))*s2)
  }
  return(t)
}
```

8.4 Code §4.3.1

8.4.1 Simulatie

```
#tijd meten
ptm<-proc.time()

#parameters
Q<-10000  #het aantal simulaties we doen om s te schatten
```

```
t<-1000  #lengte van de tijdreeksen, aantal "dagen"

#'data' waar we s voor willen schatten
N<-500 #aantal agents
Dm<-0.001 #standaard deviatie inkomende informatie stroom
lambda<-10 #marktdiepte
theta0<- matrix(0.5,1,N)
s0<-0.4

#begin model
u<-matrix(runif(t*N,0,1),t,N)
 #u(t) iid random variables uniform op [0,1], voor updaten threshold
e<-rnorm(t,0,Dm)
 #inkomende informatiestroom

theta<-matrix(0,t,N) #lege matrix voor de thresholds
phi<-matrix(0,t,N) #lege matrix voor de vraag
Z<-numeric(t) #lege vector voor excess demands
r<-numeric(t) #lege vector voor de returns

theta[1,]=theta0 #eerste rij van theta is beginverdeling thresholds

for(i in 1:N){if(e[1]< -1*theta[1,i])phi[1,i]=-1
              if(e[1]>theta[1,i])phi[1,i]=1}     #eerste rij van phi maken

Z[1]=sum(phi[1,])   #eerste waarde Z
r[1]=Z[1]/(N*lambda)#eerste waarde return

for(i in 2:t){
  for(j in 1:N){
    if(u[i,j]<s0)theta[i,j]=abs(r[i-1])
    if(u[i,j]>=s0)theta[i,j]=theta[i-1,j] #theta verder maken
    if(e[i]< -1*theta[i,j])phi[i,j]=-1 #phi daaruit maken
    if(e[i]> theta[i,j])phi[i,j]=1
  }
  Z[i]=sum(phi[i,])
  r[i]=Z[i]/(N*lambda)
}

#summery statistic voor logreturn dit model

#acf voor alle tijdsverschillen berekenen,
#en zorgen dat je kunt rekenen met die waarden
cX<-acf((r^2),plot=FALSE)$acf

#we doen alsof we niet weten met welke s deze data gegenereerd is
#Nu willen we voor deze data de beste s voor het model schatten

priorS<-runif(Q,0,1)
#vector met voor elke simulatie s uit unif(0,1), a priori verdeling
```

```
D<-matrix(0,Q,2)
#lege matrix maken om straks alle summary statistics en s'jes in op te slaan

#parameters van het model staan in de functie

#nu Q keer de gegenereerde logreturn met zijn summary statistic berekenen
# en de bijbehorende s opslaan
for(k in 1:Q){

  #s trekken uit a priori verdeling en data r (logreturns uit model) genereren
  s<-priorS[k]


  cY<-myfunction(s,t)
  z<-0
  for(l in 2:10){
    z=z + abs(cX[l]-cY[l])
  }
  # de summary statistic opslaan in de vector D samen met de bijbehorende s
  D[k,1]=z
  D[k,2]=priorS[k]

}

D<-D[order(D[,1]),]
#eerste kolom van D in oplopende volgorde zetten
#en de bijbehorende s'jes (kolom 2) mee verplaatsen

D1<-D[,2]
#vector met s'jes op volgorde van degene met kleinste 'fout' in summary statistic
#tot degene met de grootste 'fout'in summary statistic
Opt10000<-head(D1,100) #beste 100 hiervan nemen


plot(density(Opt10000),main="",col="green",xlim=c(0.3,0.55),ylim=c(0,25),xlab="",ylab="")
#verdeling beste s'jes, (=>a postriori verdeling)
mean(Opt10000)          #gemiddelde s, de 'schatting'

#totale rekentijd berekenen
proc.time()-ptm
```

8.4.2 Functie: myfunction

```
myfunction<-function(s,t){

  #parameters die verder nog nodig zijn
  N<-500 #aantal agents
  Dm<-0.001 #standaard deviatie inkomende informatie stroom
```

```
  lambda<-10 #marktdiepte
  theta0<- matrix(0.5,1,N)

  #begin model
  u<-matrix(runif(t*N,0,1),t,N)
#u(t) iid random variables uniform op [0,1], voor updaten threshold
  e<-rnorm(t,0,Dm)
#inkomende informatiestroom

  thetaf<-matrix(0,t,N) #lege matrix voor de thresholds
  phif<-matrix(0,t,N) #lege matrix voor de vraag
  Zf<-numeric(t) #lege vector voor excess demands
  rf<-numeric(t) #lege vector voor de returns

  thetaf[1,]=theta0 #eerste rij van theta is beginverdeling thresholds

  for(i in 1:N){if(e[1]< -1*thetaf[1,i])phif[1,i]=-1
                if(e[1]>thetaf[1,i])phif[1,i]=1}     #eerste rij van phi maken

  Zf[1]=sum(phif[1,])   #eerste waarde Z
  rf[1]=Z[1]/(N*lambda)#eerste waarde return

  for(i in 2:t){
    for(j in 1:N){
      if(u[i,j]<s)thetaf[i,j]=abs(rf[i-1])
      if(u[i,j]>=s)thetaf[i,j]=thetaf[i-1,j] #theta verder maken
      if(e[i]< -1*thetaf[i,j])phif[i,j]=-1 #phi daaruit maken
      if(e[i]> thetaf[i,j])phif[i,j]=1
    }
    Zf[i]=sum(phif[i,])
    rf[i]=Zf[i]/(N*lambda)
  }
  #hier eindigt het model, data r is gegenereerd


  #3 summary statistic CY voor die r berekenen
  cY<-acf((rf^2),plot=FALSE)$acf  #autocorrelaties voor r berekenen


  return(cY) #output van de functie
}
```

8.5 Code §5.3.1

8.5.1 Simulatie

```
#tijd meten
ptm<-proc.time()
```

```
#parameters en vectoren enzo

M<-500                    #het aantal tijdsstappen in de data
Q<-10^7                   #het aantal simulaties we doen om beta0 te schatten
sigma<-0.01               #sd van de ruis
sigma2<-0.05              #sd van het interval waarin we de nieuwe beta zoeken
beta0<-0.8                #beta van de data, deze willen we benaderen met de schatting
X0<-0                     #beginwaarde X
epsilon<-0.01             #marge waneer een beta is goedgekeurd
beta<-numeric(Q)          #lege vector voor de beta's

#'data' genereren die we gaan gebruiken:
# X(t) = 0.8X(t-1) + zet(t), X(0)=0, zet N(0,sigma^2) verdeeld

zetX<-rnorm(M,0,sigma) #vector z met N(0,sigma^2) verdeelde waarden voor elke t, voor X
X<-numeric(M)          #lege vector om de X'jes in te zetten straks

X[1]=X0
for(i in 2:M){
  X[i] = beta0*X[i-1] + zetX[i]
}
#summery statistic voor X berekenen
#acf voor alle tijdsverschillen berekenen,
# en zorgen dat je kunt rekenen met die waarden
cX<-acf(X,plot=FALSE)$acf
CX<-sum(cX[2:6])                      #waardes optellen voor lag 1,2,3,4 en 5
#dit is je summary statistic

#Nu doen alsof we beta0 niet meer weten en de verdeling hiervan willen benaderen

#we vinden een beta[0] mbv algoritme 2 met een dataset Y zo dat
#|CX-CY| <= epsilon
verschil<-1
while(verschil>epsilon){

  beta1=runif(1,-1,1)
  CY<-alg2(beta1,M,sigma) #alg2 geeft de samenvattingsvariabele CY
  verschil<-abs(CX-CY)    #van een dataset gegenereerd met beta terug
}
beta[1]=beta1

for(j in 2:Q){
 #genereer een beta' uit een N(beta[j-1],sigma^2) verdeling
 betaJ<-rnorm(1,beta[j-1],sigma2)

 #simuleer dataset Y' en bereken CY
 CY<-alg2(betaJ,M,sigma)
 verschil2<-abs(CX-CY)
```

```
 #genereer een u uniform op [0,1]
 u<-runif(1,0,1)

 #bepalen of je nieuwe beta' voldoet
A<-1 #die acceptatieterm
 if((verschil2<=epsilon)&(u<=A)){
  beta[j]=betaJ
 } else{
   beta[j]=beta[j-1]
 }
}

#plotten benadering a postriori verdeling
plot(density(beta),main="a postriorie dichtheid van beta",
                          xlab="waarde", ylab="dichtheid",col="red")
mean(beta)
#totale rekentijd berekenen
proc.time()-ptm
```

8.5.2 Functie: alg2

```
alg2<-function(beta,M,sigma){

  #data Y simuleren met deze beta en daar de samenvattingsvariabele van geven

  zetY<-rnorm(M,0,sigma) #vector z met N(0,sigma^2) verdeelde waarden voor elke t
  Y<-numeric(M) #lege vector om de X'jes in te zetten straks

  Y[1]=0
  for(i in 2:M){
    Y[i] = beta*Y[i-1] + zetY[i]
  }
  #summery statistic berekenen
  cY<-acf(Y,plot=FALSE)$acf
  CY<-sum(cY[2:6]) #waardes optellen voor lag 1,2,3,4 en 5

return(CY)

}
```

8.6 Code §5.4.1

```
#tijd meten
ptm<-proc.time()

#parameters
Q<-10000  #het aantal simulaties we doen om s te schatten
t<-1000  #lengte van de tijdreeksen, aantal "dagen"
sigma<-0.1 # sd van het interval waarin we nieuwe s zoeken
```

```
s<-numeric(Q) #lege vector om s in op te slaan
epsilon<-0.3 #marge wanneer s goedgekeurd is

#'data' waar we s voor willen schatten
s0<-0.4

#begin summery statistic voor logreturn dit model
cX<-myfunction(s0,t)

#we doen alsof we niet weten met welke s deze data gegenereerd is
#Nu willen we voor deze data de beste s voor het model verder contstant schatten

#we vinden een s[0] mbv algoritme 2 met een gegenereerde dataset Y
#zo dat |CX-CY|<= epsilon
verschil<-1
while(verschil>epsilon){

  s1<-runif(1,0,1)
  cY<-myfunction(s1,t)
  z<-0
  for(l in 2:10){
    z=z + abs(cX[l]-cY[l])
  }
  verschil<-z
}
s[1]=s1

for(j in 2:Q){
  #genereer s' uit uit een N(s[j-1],sigma^2) verdeling
  #en als hij buiten [0.1] ligt is hij sowieso niet
  #geaccepteerd

  sJ<-rnorm(1,s[j-1],sigma)
  if((sJ>=0)&(sJ<=1)){

  #simuleer een dataset Y' adhv s' en bereken rho(mu,muJ)
  cY<-myfunction(sJ,t)
  z<-0
  for(l in 2:10){
    z=z + abs(cX[l]-cY[l])
  }
  verschil<-z

  #genereer u uniform op [0.1]
  u<-runif(1,0,1)

  #kijken of hij geaccepteerd wordt, A is acceptatiekans
  A<-1
  if((verschil<=epsilon)&(u<=A)){
    s[j]=sJ
```

```
  } else{
    s[j]=s[j-1]
  }

  }
  else{
    s[j]=s[j-1]
  }

}

#plotten a postriori verdeling
plot(density(s),main="a postriorie dichtheid van s",xlab="waarde",
                                                    ylab="dichtheid",col="red")
mean(s)

#totale rekentijd berekenen
proc.time()-ptm
```

8.7 Code §6.2.1

```
ptm<-proc.time()

P<-14 #aantal populaties
N<-1000 #aantal kandidaten per populatie

M<-500 #het aantal tijdsstappen waar we informatie over hebben in de data
sigma<-0.01 #sd van de ruis
sigma2<-0.05 #sd van K(beta|betaX)
beta0<-0.8 #beta van de data, deze willen we benaderen met de schatting

#vector epsilon maken
epsilon<-numeric(P)
epsilon[1]=0.1
for(j in 2:P){
  epsilon[j]= 0.7*epsilon[j-1]
}

#'data' genereren die we gaan gebruiken:
# X(t) = 0.8X(t-1) + zet(t), X(0)=0, zet N(0,sigma^2) verdeeld
CX<-alg2(beta0,M,sigma) #summery statistic voor deze data

wght<-matrix((1/N),N,P) #lege matrix voor gewichten en beta's
beta<-matrix(0,N,P) #elke kolom is een populatie


for(p in 1:P){
```

```
   for(i in 1:N){

     if(p == 1){ #als p=0 betaXX uit a priori verdeling
       betaXX<-runif(1,-1,1)
     } else{ #anders genereren uit kern
       verschil<-10
       while(verschil>=epsilon[p]){ #net zo lang tot er een geaccepteerd wordt

       #pi(beta**) mag niet 0 zijn, dus betaXX in [-1,1]
       betaXX<-10
       while((betaXX<(-1))|(betaXX>1)){
       betaX<-sample(beta[,(p-1)],1,replace=TRUE,prob=wght[,(p-1)])
       betaXX<-rnorm(1,betaX,sigma2) }

       CY<-alg2(betaXX,M,sigma)
       verschil<-abs(CY-CX)
     }

     }
     beta[i,p]<-betaXX
     #gewicht berekenen van beta[i,p]
     if(p == 1){
       wght[i,p]=1
     }else{
       noemer<-0

       for(k in 1:N){
         noemer = noemer + wght[k,(p-1)]*dnorm(betaXX,beta[k,(p-1)],sigma2)
       }

       wght[i,p]=(0.5/noemer) #pi(betaXX)=0.5 voor alle betaXX (uniform)

     }

   }

   #gewichten normaliseren
   som<-sum(wght[,p]) #som van alle gewichten in een populatie
   for(l in 1:N){
     wght[l,p] = (wght[l,p]/som)
   }


}

postSMC<-beta[,P]
gewicht<-wght[,P]

#plotten a postriori verdeling
```

```
plot(density(postSMC,weight = gewicht),main="a postriorie dichtheid van beta",
     xlab="waarde", ylab="dichtheid",col="purple",xlim=c(0.65,0.9),ylim=c(0,14))
weighted.mean(postSMC,gewicht)

proc.time()-ptm
```

8.8 Code §6.3.1

```
ptm<-proc.time()


P<-10 #aantal populaties
N<-1000 #aantal kandidaten per populatie

t<-1000  #lengte van de tijdreeksen, aantal "dagen"
sigma<-0.1 # sd van het interval waarin we nieuwe s zoeken

#vector epsilon maken
epsilon<-numeric(P)
epsilon[1]=0.45
for(j in 2:P){
  epsilon[j]= 0.9*epsilon[j-1]
}

#'data' waarvoor we s willen schatten
#met bijbehorende summary statisticsvector
s0<-0.4
cX<-myfunction(s0,t)

#nu deze s0 schatten voor het model verder contstant
wght<-matrix((1/N),N,P) #matrices voor gewichten en beta's
s<-matrix(0,N,P) #elke kolom is een populatie

for(p in 1:P){

  for(i in 1:N){

    if(p == 1){ #als p=0 sXX uit a priori verdeling
      sXX<-runif(1,0,1)
    } else{ #anders genereren uit kern
      verschil<-10
      while(verschil>=epsilon[p]){ #net zo lang tot er een geaccepteerd wordt

        #pi(s**) mag niet 0 zijn, dus sXX in [0,1]
        sXX<-10
        while((sXX<0)|(sXX>1)){
          sX<-sample(s[,(p-1)],1,replace=TRUE,prob=wght[,(p-1)])
          sXX<-rnorm(1,sX,sigma) }
```

```
        cY<-myfunction(sXX,t)
        #verschil in samenvattingsvariabelen berekenen
         z<-0
         for(l in 2:10){
              z=z + abs(cX[l]-cY[l])
          }
          verschil<-z
      }

    }
    s[i,p]<-sXX
    #gewicht berekenen van s[i,p]
    if(p == 1){
      wght[i,p]=1
    }else{
      noemer<-0

      for(k in 1:N){
        noemer = noemer + wght[k,(p-1)]*dnorm(sXX,s[k,(p-1)],sigma)
      }

      wght[i,p]=(1/noemer) #pi(sXX)=1 voor alle sXX (uniform)

    }

  }

  #gewichten normaliseren
  som<-sum(wght[,p]) #som van alle gewichten in een populatie
  for(l in 1:N){
    wght[l,p] = (wght[l,p]/som)
  }


}

postSMC<-s[,P]
gewicht<-wght[,P]

plot(density(postSMC,weight = gewicht),main="a postriorie dichtheid van s",
    xlab="waarde", ylab="dichtheid",col="red")#,xlim=c(0.65,0.9),ylim=c(0,14))
weighted.mean(postSMC,gewicht)

proc.time()-ptm
```

Bibliografie

[1] Francois Ghoulmie, Rama Cont, Jean-Pierre Nadal
24 maart 2005
stacks.iop.org/JPhysCM/17/S1259

[2] Financial risk forecasting
Jon Danielsson
ISBN-13: 978-0470669433

[3] Monte Carlo Methods in Financial Engineering
Paul Glasserman
ISBN: 0-387-00451-3

[4] Introduction to Time Series and Forecasting, Second Edition
Peter J. Brockwell, Richard A. Davis
ISBN 0-387-95351-5

[5] Page 286, Mathematical Statistics and Data Analysis, Third Edition
John A. Rice
ISBN 0-495-11868-0

[6] Approximate Bayesian Computational methods
Jean-Michel Marin, Pierre Pudlo, Christian P. Robert, Robin J. Ryder
30 mei 2011
arXiv:1101.0955v2 [stat.CO]

[7] `https://en.wikipedia.org/wiki/File:Approximate_Bayesian_computation_conceptual_overview.svg`

[8] Understanding the Metropolis-Hastings Algorithm
Siddhartha Chib, Edward Greenberg
The American Statistician, November 1995, Vol. 49, No 4

[9] Continuity, Integration and Fourier Theory
A.C. Zaanen
ISBN 978-3-540-50017-9

[10] Tutorial on ABC rejection and AMC SMC for parameter estimation and model selection
Tina Toni, Michael P. H. Stumpf
19 januari 2010
arXiv:0910.4472v2 [stat.CO]

[11] Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems
Tina Toni, David Welch, Natalja Strelkowa, Andreas Ipsen, Michael P.H. Strumpf
J.R. Soc. Interface 2009 6, doi: 10.1098/rsif.2008.0172, 6 februari 2009