Hamiltonin Monte Carlon soveltaminen finanssiaikasarjoihin

Tilastotieteen pro gradu -tutkielma

30.10.2018

Tero Lähderanta

Matematiikan ja tilastotieteen laitos

Jyväskylän yliopisto

Tiivistelmä

Markovin ketju Monte Carlo -menetelmät ovat olleet tärkeä osa Bayes-tilastotiedettä jo 90-luvulta saakka. Monet perinteiset MCMC-algoritmit, kuten Metropolis-algoritmi ja Gibbsin otanta, ovat yhä suuressa suosiossa tutkijoiden keskuudessa. Nämä yksinkertaiset simulaatioalgoritmit muuttuvat sitä tehottomammiksi, mitä monimutkaisemmista malleista on kysymys. Tässä tutkielmassa esitellään Hamiltonin Monte Carlo, jolla pyritään ratkaisemaan monimutkaisten mallien ongelman simuloinnissa. HMC-algoritmin matemaattisen haastavuuden takia algoritmin toiminta esitetään ensin yksinkertaisten esimerkkien kautta, minkä jälkeen syvennytään sen rakenteeseen ja teoreettiseen taustaan. Tämän lisäksi vertaillaan HMC:n ja Metropolis-algoritmin tehokkuutta ja autokorrelaatioita kahdessa finanssimallissa samalla käyden läpi algoritmin implementoinnin haasteet.

Esimerkinomaisena sovelluskohteena käytetään kahta finanssimallia, joiden avulla mallinnetaan osake- ja korkosijoitusten tuottoa. Bayesiläinen lähestymistapa on luonteva tapa arvioida finanssimallien parametrien epävarmuutta. Molemmissa valituissa malleissa HMC osoittautui ajallisesti hitaammaksi kuin Metropolis-algoritmi: samankaltaisten tulosten saaminen vaati HMC-algoritmissa huomattavasti vähemmän iteraatioita kuin Metropolis-algoritmissa, mutta yksittäisen arvon generoiminen oli HMC:ssä huomattavasti hitaampaa. HMC-algoritmin tuottaman ketjun jäsenten välinen autokorrelaatio oli kuitenkin merkittävästi pienempää mitä Metropolis-algoritmissa.

Sisältö

1 Johdanto

Markovin ketju Monte Carlo -menetelmät ovat tärkeässä osassa Bayes-tilastotieteen inferenssissä, jossa käytetään edelleen paljon perinteisiä simulointialgoritmejä, kuten Metropolis-algoritmia ja Gibbsin otantaa. Nämä perinteiset menetelmät jäävät kuitenkin tehottomiksi mallien muuttuessa yhä monimutkaisemmiksi. Vuodesta 1995 alkaen uusia tehokkaampia menetelmiä on kehitelty lukuisia: esimerkiksi Particle Filtering (Del Moral, 1996) ja adaptiivinen MCMC (Andrieu & Thoms, 2008). Voidaankin ajatella, että Metropolis-algoritmi ja Gibbsin otanta ovat rakennuspalikoita uusille kehittyneemmille simulointialgoritmeille (Gelman et al., 2014, s. 280-281).

Yksi simuloinnin kannalta tärkeä algoritmi on Hamiltonin Monte Carlo (HMC), joka perustuu vahvasti Metropolis-algoritmiin. Kuten Metropolis-algoritmissa, myös HMC:ssä generoidaan seuraava Markovin ketjun ehdotus, joka hyväksytään tietyllä todennäköisyydellä. HMC:ssä ehdotuksen generoimista parannetaan hyödyntämällä posteriorijakauman gradienttia. Tämä mahdollistaa sen, että algoritmi liikkuu nopeammin kohdejakauman läpi ja hyväksyttyjen ehdotusten osuus on suurempi kuin Metropolis-algoritmissa. Monimutkaisissa asetelmissa menetelmä on osoittautunut nopeammaksi ja luotettavammaksi kuin perinteiset Markovin ketjua hyödyntävät algoritmit (Gelman et al., 2014, s. 300). Tilastotieteessä HMC:n matemaattinen haastavuus on vaikuttanut algoritmin suosioon negatiivisesti: teoreettisesti menetelmä perustuu differentiaaligeometriaan, joka ei välttämättä ole tuttu menetelmä tilastotieteen tutkijalle (Betancourt, 2017).

Tässä työssä esitellään Hamiltonin Monte Carlo -algoritmi ja vertaillaan sitä yksinkertaisempiin algoritmeihin. Toisessa pääluvussa esitellään Hamiltonin Monte Carlo -algoritmin perusidea yksinkertaisten esimerkkien kautta. Tämän lisäksi syvennytään algoritmin askeliin ja hienosäätöparametreihin teoreettisemmasta näkökulmasta. Kolmannessa luvussa esitellään sovelluskohteena toimivat kaksi finanssimallia, joiden aineistona on käytetty korko- ja osakeindeksisarjoja. Neljännessä pääluvussa sovelletaan Hamiltonin Monte Carloa ja Metropolis-algoritmia kahteen monimutkaiseen finanssimalliin, ja tämän perusteella tarkastellaan HMC:n tehokkuutta suhteessa yksinkertaisempiin algoritmeihin. Lisäksi pohditaan, mitä tulee ottaa huomioon sovellettaessa HMC:tä hierarkkisiin malleihin. Viidennessä pääluvussa pohditaan HMC:n hyviä ja huonoja puolia ja eritellään mahdollisia jatkotutkimusaiheita.

2 Markovin ketju Monte Carlo -menetelmät

Markovin ketju Monte Carlo -menetelmät (MCMC) saivat alkunsa jo toisen maailmansodan aikana, vaikka Bayes-tilastotieteessä menetelmät yleistyivätkin vasta 1990-luvulla. Metropolis-algoritmi julkaistiin ensimmäisenä MCMC-menetelmänä vuonna 1953 fysiikan alan artikkelissa, jossa ratkaistiin integraali liittyen partikkelien liikkeeseen kaasussa (Metropolis et al., 1953). Esimerkiksi kuva-analyysissa, spatiaalisessa tilastotieteessä sekä fysiikan aloilla MCMC-menetelmät alkoivat tämän jälkeen hiljalleen yleistyä. Bayes-tilastotieteessä MCMC yleistyi kuitenkin vasta 1990-luvun alussa Gelfandin ja Smithin artikkelin (Gelfand & Smith, 1990) sekä tietokoneiden kehittymisen myötä. Tästä alkoi ns. Bayes-vallankumous, jonka myötä kehitettiin lukuisia uusia MCMC-algoritmeja. (Robert & Casella, 2011)

Hamiltonin Monte Carlo -algoritmi (HMC) perustuu Hamiltonin mekaniikkaan, joka juontaa juurensa William Rowan Hamiltonin tutkimukseen *On General Method in Dynamics* vuodelta 1834 (Abraham & Marsden, 1978). Hamiltonin mekaniikka on vakiinnuttanut paikkansa yhtenä lähestymistapana klassiseen mekaniikkaan (Bell, 1963). Varsinainen HMC-algoritmi esiteltiin ensimmäisen kerran 1980-luvulla julkaistussa fysiikan alan artikkelissa, jossa käsiteltiin kvanttikromodynamiikan laskentaongelmia (Duane et al., 1987). Tässä vaiheessa menetelmä tosin tunnettiin vielä nimellä Hybrid Monte Carlo. Menetelmää hyödynnettiin tilastotieteessä ensimmäistä kertaa vasta vuonna 1995, jolloin Radford Neal hyödynsi algoritmia bayesiläisissä neuroverkoissa (Neal, 1995). Ensimmäisen kerran nimitystä Hamiltonin Monte Carlo käytettiin MacKeyn oppikirjassa vuonna 2003 (MacKey, 2003). Vasta vuonna 2011 menetelmä tuli yleiseen käyttöön tilastotieteessä Nealin artikkelin (Neal, 2011) myötä. HMC:n pohjalta on kehitetty tietokoneohjelma Stan (Sampling through adaptive neighborhoods) (Stan development team, 2017), joka soveltaa automaattisesti Hamiltonin Monte Carloa annettuun malliin.

Tässä luvussa perehdytään Bayes-tilastotieteen perusteisiin ja esitellään MCMC-menetelmät, jotka ovat tärkeä osa modernia Bayes-inferenssiä. Tämän jälkeen tarkastellaan tässä työssä vertailualgoritmina toimivaa Metropolis-algoritmia, joka on vakiinnuttanut paikkansa Bayes-inferenssin tärkeimpänä työkaluna. Lopuksi esitellään Metropolis-algoritmiin pohjautuva Hamiltonin Monte Carlo. HMC-algoritmia tarkastellaan ensin yksinkertaisten esimerkkien kautta ja tämän jälkeen syvennytään algoritmin toimintaan teoreettisemmal-

ta kannalta.

2.1 Bayes-inferenssi

Bayes-tilastotieteen inferenssissä sovitetaan todennäköisyysmalli dataan y, ja mallin parametrien θ tai muiden havaitsemattomien arvojen, kuten ennusteiden (eng. *predictions*), tulokset esitetään todennäköisyysjakaumien avulla. Tästä johtuen Bayes-menetelmien ydin on käyttää inferenssin epävarmuuden kuvailussa todennäköisyyksiä, jotka ovat ehdollisia havaintojen y suhteen. Tarkemmin kuvailtuna inferenssi perustuu posteriorijakaumaan $p(\theta|y)$, eli tuntemattomien jakaumaan ehdolla havainnot (Gelman et al., 2014, s. 7):

$$p(\theta|y) = \frac{p(\theta)p(y|\theta)}{p(y)}, \text{ missä } p(y) = \int ... \int p(\theta)p(y|\theta)d\theta, \text{ kun } \theta \text{ oletetaan jatkuvaksi.}$$

Inferenssissä tutkittavaan aineistoon sovitetaan malli $p(y|\theta)$, joka on havainnon jakauma ehdolla, että θ oletetaan kiinnitetyksi. Nämä parametrit ovat tuntemattomia eli niihin liittyy epävarmuutta. Tätä epävarmuutta mallinnetaan parametrien priorijakaumalla $p(\theta)$, jolla on tärkeä rooli inferenssissä. Priorijakauman valinnalla voidaan ottaa huomioon tutkijan ennakkokäsityksiä ja tietoa kyseisestä ilmiöstä. Posteriorijakaumassa yhdistyy siis tutkijan ennakkotiedot priorijakauman muodossa ja aineiston informaatio mallin avulla. Priori voi olla informatiivinen, jos ennakkotieto on vahva, mutta usein prioriksi valitaan kuitenkin epäinformatiivinen priori, jolloin priorin valinnalla ei ole suurta merkitystä päätelmiin. Tavallisesti parametrit θ noudattavat jotain jakaumaa ja tämän jakauman parametreja kutsutaan hyperparametreiksi. Tämä hierarkkinen ajattelu on usein hyödyllinen keino mallintaa aineistoa. (Gelman et al., 2014, s. 101, Box & Tiao, 1992). Bayes-inferenssin tarkempi kuvailu ja havainnollistavia esimerkkejä löytyy mm. kirjoista Gelman et al. (2014) ja Box & Tiao (1992).

Havainnon reunajakauman $p(y)$ analyyttinen laskeminen on usein mahdotonta, sillä tavallisesti θ on moniulotteinen, jolloin reunajakauman laskemiseksi pitäisi ratkaista moniulotteinen integraali. Yhden tai kahden parametrin tapauksissa integraali voidaan laskea analyyttisesti, mutta esimerkiksi hierarkkisissa malleissa parametreja on usein enemmän. Ongelman kiertämiseksi on useita eri ratkaisuja, kuten konjugaattipriorit, erilaiset posteriorin approksimaatiot ja posteriorin simuloiminen.

Mahdollinen ratkaisu posteriorin laskemiseen on konjugaattipriorien käyttö. Tällöin pa-

rametrin prioriksi valitaan sitä vastaavan uskottavuusfunktion konjugaattipriori, mikä takaa sen, että saatu posteriorijakauma kuuluu samaan jakaumaperheeseen kuin priori. Jos malliksi valitaan esimerkiksi Poisson-jakauma, niin tällöin intensiteettiparametrin konjugaattipriori on Gamma-jakauma, jolloin posteriorijakauma on myös Gamma-jakauma. Menetelmän avulla posteriorijakauma saadaan laskettua tarkasti, mutta merkittävä ongelma on, että valittu konjugaattipriori ei välttämättä kuvaa ennakkotietoja todenmukaisesti. Ratkaisu ei myöskään sovellu tilanteisiin, jossa tarkastellaan monimutkaisia hierarkkisia malleja. (Diaconis & Ylvisaker, 1979)

Toinen vaihtoehto on posteriorijakauman approksimoiminen jollain tunnetulla yksinkertaisella jakaumalla, kuten normaalijakaumalla, mikäli posteriorijakauma on yksimodaalinen ja lähes symmetrinen. Tällöin saadusta posteriorin approksimaatiosta voidaan laskea yksittäisten parametrien reunaposteriorijakaumat integroimalla kyseistä jakaumaa. Tavallisesti Bayes-analyysissa tarkastellaan posteriorin logaritmia, jolloin normaalijakauma-approksimaatiossa tarkastellaan parametrien θ toisen asteen polynomifunktiota. (Gelman et al., 2014, s. 83-84)

2.2 Simulointialgoritmit

Tietokoneiden kehittymisen myötä erilaisia simulaatiomenetelmiä käytetään ratkaisemaan posteriorijakauman laskemiseen liittyviä ongelmia (Robert & Casella, 2011). Suuri syy simulaatioalgoritmien suosioon Bayes-tilastotieteessä on niiden toimivuus erilaisten mallien kanssa moniulotteisissakin tilanteissa. Näiden stokastisten menetelmien ideana on simuloida posteriorijakaumaa eli tuottaa posteriorijakaumasta havaintoja θ_i, missä $i = 1, 2, 3, ..., M$. Tilastollinen päättely perustuu tähän otokseen ja jonkin funktion $f(\theta)$ odotusarvon estimoimiseen (Gelman et al., 2014, s. 262):

$$E(f(\theta)|y) = \int f(\theta)p(\theta|y)d\theta \approx \frac{1}{M}\sum_{i=1}^{M} f(\theta_i).$$

Simulointimenetelmät perustuvat vahvasti stokastisiin prosesseihin. Stokastinen prosessi on joukko $X_t|t \in T$, missä X_t ovat satunnaismuuttujia hetkellä t. Bayes-tilastotieteessä kiinnostuksen kohteena ovat mallin parametrit θ ja tällöin stokastinen prosessi on muotoa $\theta_t|t \in T$.

Simulointimenetelmät voidaan jakaa kahteen osa-alueeseen havaintojen riippuvuuden

mukaan: tavallisilla Monte Carlo -menetelmillä tuotetaan riippumattomia havaintoja posteriorijakaumasta, kun taas Markovin ketju Monte Carlo -menetelmillä tuotetaan riippuvia havaintoja. Ilmaisu *Monte Carlo* viittaa menetelmiin, jossa satunnaisotosten avulla saadaan tietoa kiinnostavasta jakaumasta. Yksi tavallisimmista Monte Carlo -menetelmistä on tärkeysotanta, jossa simuloidaan jotain tunnettua jakaumaa $g(\theta)$, joka approksimoi posterioria $p(\theta|y)$. Kyseinen menetelmä perustuu normeeraamattomaan posterioriin, jolloin integraalia $p(y)$ ei tarvitse laskea. (Gelman et al., 2014, s. 265-266)

Menetelmät, jossa tuotetaan riippumattomia havaintoja posteriorijakaumasta, toimivat hyvin monissa tilanteissa, mutta niitä ei voida pitää geneerisinä algoritmeina. Monikäyttöisin menetelmä useimmille malleille on riippuvien havaintojen tuottaminen posteriorijakaumasta Markovin ketju Monte Carlo -menetelmällä (MCMC). MCMC-menetelmät ovat kokoelma algoritmeja, joiden avulla voidaan simuloida arvoja halutusta todennäköisyysjakaumasta muodostamalla Markovin ketju, jonka tasapainojakauma on kiinnostava jakauma eli posteriori (Metropolis & Ulam, 1949). Tavallisesti simuloitava jakauma on jatkuva, mutta yksinkertaisuuden vuoksi tarkastellaan Markovin ketjua, jolla on diskreetti tila-avaruus. Markovin ketju on diskreettiaikainen stokastinen prosessi, jossa uusi tila X_{n+1} riippuu ainoastaan edellisestä tilasta:

$$P(X_{n+1} = x|X_1 = x_1, X_2 = x_2, ..., X_n = x_n) = P(X_{n+1} = x|X_n = x_n), \text{ kun } P(X_1 = x_1, ..., X_n = x_n) > 0$$

Markovin ketju suppenee tietyin oletuksin, jolloin ketjun arvot muodostavat otoksen posteriorista tietyn askelmäärän jälkeen. Toisin sanoen: kun $t \to \infty$, niin satunnaismuuttuja θ_t jakauma suppenee kohti tasapainojakaumaa, mikä ei riipu Markovin ketjun alkuarvosta θ_0. Tavallisesti MCMC-analyysissa muodostetaan useita ketjuja θ^1, θ^2, θ^3,... ja jokaiselle ketjulle on omat parametrien alkuarvot: θ_0^1, θ_0^2, θ_0^3,... Uusi Markovin ketjun arvo θ_t simuloidaan siirtymäjakaumasta $T_t(\theta_t|\theta_{t-1})$, joka riippuu edellisestä ketjun arvosta θ_{t-1}. Siirtymäjakauma muodostetaan siten, että Markovin ketju konvergoi kohti posteriorijakaumaa.

MCMC-menetelmän toiminnallisuuden kannalta oleellisin seikka on Markovin ketjun tasapainojakauma. Kun tasapainojakauma on sama kuin kiinnostava jakauma eli posteriori, niin MCMC-menetelmän tuloksena saatu Markovin ketju on likipitäen otos kyseisestä jakaumasta. Tasapainojakauman konstruoiminen perustuu simulointialgoritmeissa lokaaliin

tasapainoehtoon. Merkitään, että S on tarkasteltavan Markovin ketjun tila-avaruus, ketjun tasapainojakauma on $\pi_i | i \in S$ ja q_{ij} on todennäköisyys siirtyä tilasta i tilaan j. Lokaali tasapainoehto on tällöin:

$$\pi_i q_{ij} = \pi_j q_{ji}, \text{ kaikilla } i, j \in S.$$

Markovin ketjun muodostamisen jälkeen tarkistetaan ketjun konvergoiminen tavallisesti tarkastelemalla otospolkukuvia ja kertoimen $\hat{R}$ arvoja. Markovin ketjussa on tavallisesti niin sanottu burn in -jakso, mikä jätetään lopullisesta tarkastelusta pois. Tämä johtuu siitä, että algoritmin ensimmäisillä askelilla Markovin ketjun arvot eivät välttämättä tule halutusta jakaumasta, koska tutkijan asettamat alkuarvot ovat kaukana jakauman tyypillisistä arvoista. Merkitään burn in -jakson pituutta n_{burnin}, jonka arvo on tilanteesta riippuva. Burn in -jakson pituutta voidaan tarkastella otospolkukuvien avulla.

MCMC-menetelmien etu on suuri verrattuna konjugaattipriorireihin tai integraalin numeeriseen laskemiseen ja siksi MCMC on vakiinnuttanut paikkansa Bayes-tilastotieteessä. MCMC-menetelmät toimivat useimmissa tilanteissa moitteettomasti ja ne antavat vapauden tarkastella lähes minkälaisia malleja ja prioreja tahansa, toisin kuin esim. konjugaattipriorien tapauksessa, missä priori valitaan aina tietystä jakaumaperheestä käytettävän mallin mukaan. Kirjava joukko erilaisia MCMC-algoritmeja varmistaa myös mahdollisuuden optimoida algoritmin toimintaa mallista riippuen: esimerkiksi Metropolis-algoritmi soveltuu erinomaisesti alle viiden parametrin malleille, kun taas Hamiltonin Monte Carloa voidaan käyttää tilanteissa, missä parametreja on paljon ja ne ovat selkeästi riippuvia toisistaan (Neal, 2010). Yhdistelemällä yksinkertaisia algoritmeja muiden algoritmien kanssa voidaan muodostaa erilaisiin malleihin soveltuvia algoritmiyhdistelmiä (Gelman et al., 2014, s. 275-290).

MCMC-algoritmien käyttöön liittyy monenlaisia haasteita. Etenkin monimutkaisten mallien kohdalla halutun tarkkuuden saavuttaminen vaatii useita iteraatioita, jolloin simulaatioon menee enemmän aikaa. Perinteisten MCMC-menetelmien kohdalla haasteita saattaa ilmetä esimerkiksi autokorrelaatiossa ja sekoittumisessa, kun parametrien määrä kasvaa. Ongelmaa voidaan yrittää ratkaista optimoimalla algoritmin parametreja, mutta mitä moniulotteisempia malleja tarkastellaan, sitä hitaammaksi Metropolis-algoritmi muuttuu (Gelman et al., 2014, s. 300). Metropolis-algoritmin haasteita käsitellään lisää luvussa 2.3.

2.3 Metropolis-algoritmi

Metropolis-algoritmi on yksi tärkeimmistä, menestyneimmistä ja yksinkertaisimmista Monte Carlo -simulaatiomenetelmistä (Beichl & Sullivan, 2000). Käytännöllisyytensä takia Metropolis-algoritmi on edelleen laajassa käytössä Bayes-tilastotieteessä ja monet uudemmat algoritmit perustuvat samaan rakenteeseen.

Metropolis-algoritmin yksi kierros voidaan jakaa kahteen osaan: (1) uuden ehdotuksen θ_* generoimiseen ehdotusjakaumasta ja (2) hyväksymis/hylkäämis -vaiheeseen. Vaiheessa 1 ehdotus θ_* arvotaan ehdotusjakaumasta, joka Metropolis-algoritmin tapauksessa on symmetrinen jakauma. Jälkimmäisessä vaiheessa lasketaan hyväksymistodennäköisyys α, joka kertoo, millä todennäköisyydellä kyseinen ehdotus θ_* hyväksytään. Hyväksymistodennäköisyyden laskemiseen tarvitaan ainoastaan tiheysfunktion arvot pisteessä θ_* ja edellisessä ketjun pisteessä θ_t, joten posteriorin kaavassa esiintyvän integraalin laskeminen voidaan välttää. Mikäli uusi ehdotus θ_* hyväksytään kierroksella k, niin asetetaan $\theta_k = \theta_*$. Algoritmi etenee seuraavasti (Gelman et al., 2014, s. 278):

1. Valitaan parametrien alkuarvot θ_0. Alkuarvot voivat olla joko tutkijan karkea arvio tai tarkempi approksimaatio posteriorijakaumasta.

2. Seuraava käydään läpi iteraatioilla $i = 1, 2, 3, ..., N$.

   (a) Generoidaan uusi ehdotus θ_* ehdotusjakaumasta. Metropolis-algoritmin tapauksessa ehdotusjakauma on symmetrinen.

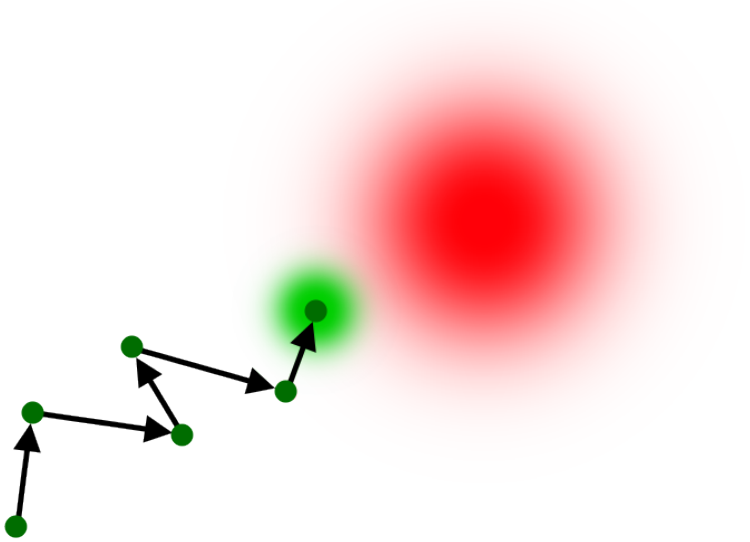
   (b) Lasketaan normeeraamattomien posteriorijakaumien suhde:

   $$r = \frac{p(\theta_*|y)}{p(\theta_{i-1}|y)}.$$

   Asetetaan $\theta_i = \theta_*$ todennäköisyydellä $min(r, 1)$. Muissa tapauksissa asetetaan $\theta_i = \theta_{i-1}$.

Algoritmin tuloksena saadaan N simuloitua arvoa $\theta_0, \theta_1, \theta_2, ..., \theta_N$, joita käytetään posteriorijakauman kuvailuun. Kuviossa 1 havainnollistetaan Metropolis-algoritmin toimintaa kaksiulotteisessa tilanteessa.

Metropolis-algoritmin tehokkuutta voidaan parantaa optimoimalla ehdotusjakauman parametrit. Tavallisesti ehdotusjakaumaksi valitaan multinormaalijakauma, jonka odotusarvo on Markovin ketjun tämänhetkinen piste θ_t ja optimaalinen kovarianssimatriisi



Kuvio 1: Hahmotelma Metropolis-algoritmin etenemisestä kaksiulotteisessa tilanteessa. Punainen alue kuvaa posteriorijakaumaa ja vihreä alue algoritmin ehdotusjakaumaa. Vihreät ympyrät ovat hyväksyttyjä ehdotuksia eli algoritmin simuloidut arvot

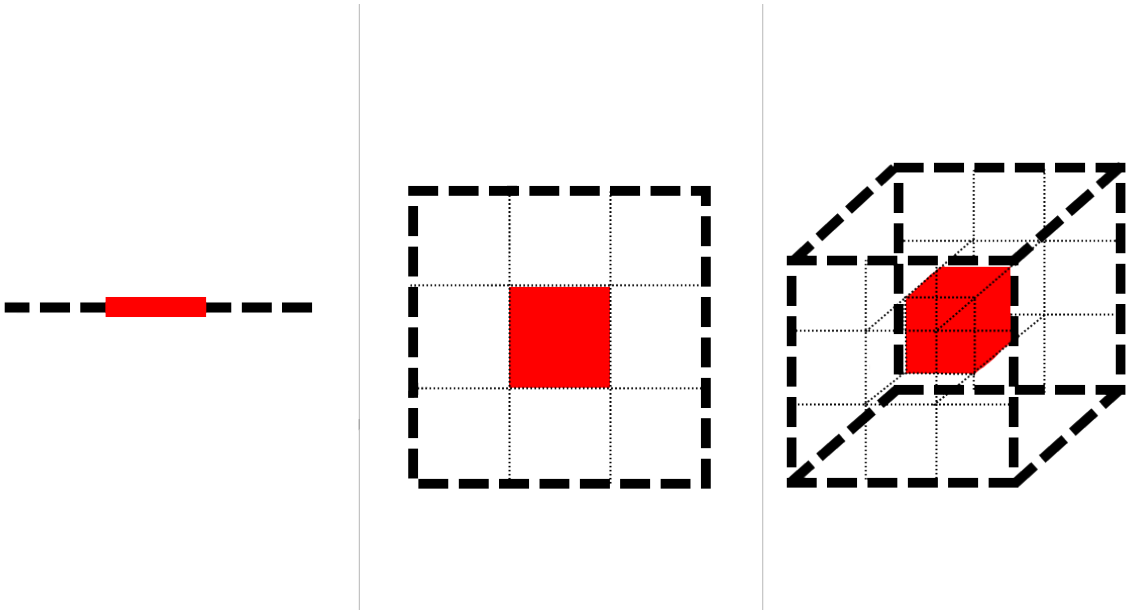
on noin $(2.4/\sqrt{d})^2\Sigma$, missä d on mallin parametrien määrä ja matriisi Σ on approksimaatio jakauman varianssista. (Gelman et al., 2014, s. 296)

Metropolis-algoritmi pitäisi toimia millä tahansa symmetrisellä ehdotusjakaumalla, mutta käytännössä algoritmin tehokkuus kärsii, kun ehdotusjakauma on huonosti valittu: jos ehdotusjakauma on liian leveä, niin uusi ehdotus θ_* tulee todennäköisemmin kiinnostavan alueen ulkopuolelta. Jos taas ehdotusjakauma on liian kapea, niin ehdotuksia hyväksytään todennäköisemmin, mutta Markovin ketjun arvot ovat niin lähellä toisiaan, että algoritmi liikkuu hitaasti. (Gelman et al., 2014, s. 300)

2.4 Moniulotteiset todennäköisyysjakaumat

Monet modernit tilastolliset mallit sisältävät ongelmasta riippuen useita toisistaan riippuvia parametreja ja nämä parametrit muodostavat erilaisia rakenteita. Tällainen hierarkkinen ajattelu auttaa ymmärtämään tilastollisen ongelman luonnetta ja asioiden syy-seuraus -suhteita. Monissa tilanteissa ei-hierarkkiset mallit eivät sovi hierarkkiseen aineistoon. Hierarkkiset mallit lisäävät parametrien määrää ja tekevät siten posteriorijakaumasta monimutkaisemman. Moniulotteisten jakaumien geometria on usein ongelma simulaatioalgoritmien tehokkuuden kannalta. (Gelman et al., 2014, s. 101)

Kuviosta 2 huomataan, että minkä tahansa tilan ulkopuolella oleva tilavuus on suurempi kuin sisäpuolella oleva tilavuus. Mitä moniulotteisempaa avaruutta tarkastellaan, sitä suurempi näiden tilavuuksien ero on. Esimerkiksi kuviossa 2 yksiulotteisessa avaruudessa punaisella merkityn tilan osuus on kolmannes koko tilasta. Jo kaksiulotteisessa avaruudessa ympäröivän alueen koko on kahdeksankertainen verrattuna punaiseen alueeseen ja kolmiulotteisessa tilanteessa 26-kertainen. Kun parametreja lisätään, niin ulkopuolella olevan tilavuuden osuus kasvaa räjähdysmäisesti: yleisesti kun tarkastellaan D-ulotteista avaruutta, niin ulkopuolella oleva tilavuus on $3^D - 1$ kertainen punaiseen alueeseen nähden. (Betancourt, 2017)



Kuvio 2: Esimerkki dimension kasvattamisen vaikutuksesta tyhjän alueen osuuteen. Vasemmalla punaisen alueen osuus on $\frac{1}{3}$, keskellä $\frac{1}{9}$ ja oikealla $\frac{1}{27}$.

Merkitään kohdejakaumaa eli tässä tapauksessa posterioria π, parametriavaruutta Q ja dimensiota D. Oleellinen osa Bayes-inferenssiä on jonkin funktion f odotusarvon laskeminen:

$$E_\pi(f) = \int_Q \pi(q) f(q) dq.$$

Koska odotusarvoa ei voida tavallisessa tilanteessa laskea analyyttisesti, käytetään niiden approksimaatioon erilaisia algoritmeja, joiden tarkkuus riippuu täysin laskennallisesta tehosta. Siksi on erityisen tärkeää, että laskentakapasiteetti käytetään arvoihin, joilla

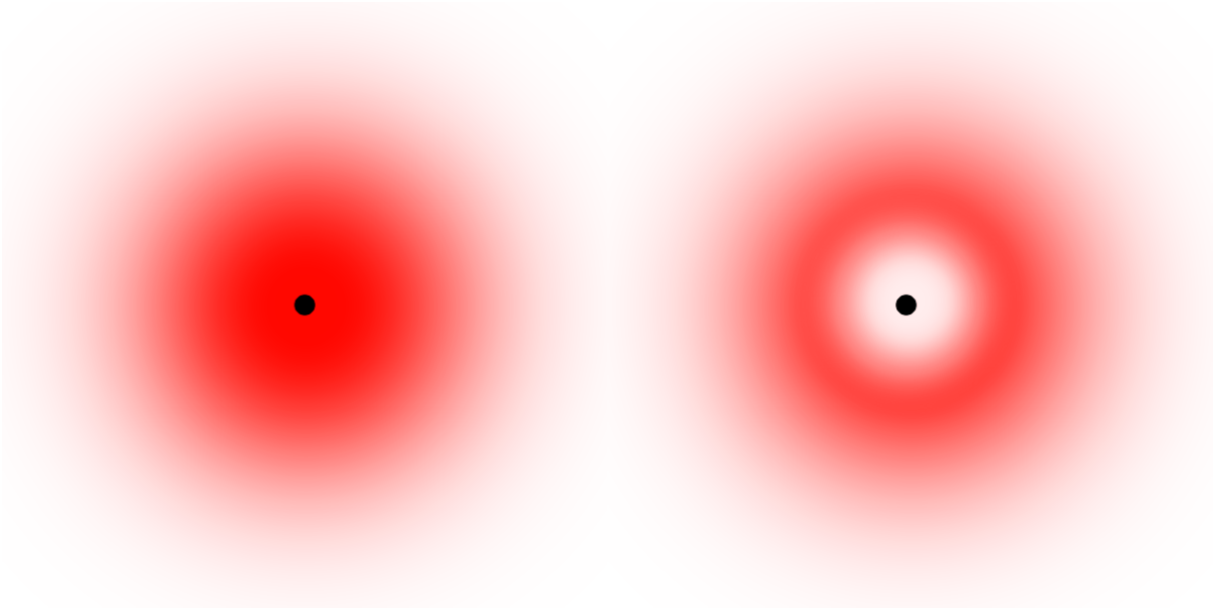
on merkitystä odotusarvon laskemisessa; toisin sanoen keskitytään arvoihin, joissa integrandi $\pi(q)f(q)$ on suurimmillaan. Tämän intuition perusteella olisi järkevintä tarkastella aluetta, missä jakauma π ja funktio f saavat suurimmat arvonsa. On turhaa käyttää laskentatehoa arvoihin, joilla on hyvin pieni vaikutus integraalin arvoon. Ongelma korostuu erityisesti moniulotteisten jakaumien kohdalla. (Betancourt, 2017)

Tätä voidaan havainnollistaa tyypillisen joukon käsitteellä (eng. *typical set*). Tyypillisellä joukolla tarkoitetaan niiden pisteiden joukkoa, joilla on suurin vaikutus odotusarvoon. Laskentatehon kannalta olisi siis järkevintä, että algoritmissa käytettäisiin mahdollisimman paljon tyypillisen joukon pisteitä. Tyypillisen joukon käsitteen hahmottamiseksi on tärkeää ymmärtää sekä jakauman tiheysfunktion arvon muutokset että tilavuuden ominaisuudet moniulotteisessa avaruudessa. Tiheysfunktion arvolla $\pi(q)$ on luonnollisesti suuri merkitys integraalin arvoon, jolloin moodin lähellä olevalla tilalla on suuri vaikutus odotusarvoon. On kuitenkin huomioitava, että myös tilavuus dq vaikuttaa tyypillisen alueen sijaintiin: vaikka tiheys on suurimmillaan moodin lähellä, pienen tilavuuden takia se ei juuri vaikuta odotusarvoon. Tästä johtuen tyypillinen alue ei sijaitse suoraan jakauman moodissa, vaan sen läheisyydessä (kuvio 3). (Betancourt, 2017).

Useimmat Monte Carlo -algoritmit eivät ota tyypillistä aluetta huomioon, joten niiden tehokkuus laskee huomattavasti. MCMC-menetelmät kuitenkin perustuvat siihen, että ne generoivat pisteitä ainoastaan tyypilliseltä alueelta. Esimerkiksi Metropolis-algoritmin ehdotusjakauma ehdottaa pisteitä kaukaa moodista, koska tilavuus on tällä alueella suurempi. Posteriorijakaumien suhteesta laskettu hyväksymistodennäköisyys taas varmistaa, että arvot tulevat tyypilliseltä alueelta, eivätkä ns. tyhjästä tilasta. (Betancourt, 2017)

Mitä enemmän parametreja tarkasteltavassa mallissa on, sitä tehottomammaksi Metropolis-algoritmi muuttuu. Moniulotteisten jakaumien tapauksessa on tavallista, että hyväksymistodennäköisyys on häviävän pieni, mikä johtuu siitä, että tyhjän tilan osuus on hyvin suuri (kuvio 2) ja suurin osa uusista ehdotuksista tulee tästä tyhjästä tilasta. Tällöin uusi ehdotus hylätään tavallista useammin, joten Markovin ketju liikkuu hyvin harvoin. Ongelmaa voidaan yrittää korjata esimerkiksi kaventamalla ehdotusjakaumaa, jolloin hyväksymistodennäköisyys suurenee. Tämä johtaa kuitenkin siihen, että Markovin ketju liikkuu pienin askelin ja simulointi etenee hyvin hitaasti. Tässä tilanteessa myös autokorrelaatiot ovat liian suuria. Pahimmassa tapauksessa estimaattorit ovat harhaisia,

sillä Markovin ketju ei ole käynyt tyypillistä aluetta kokonaan läpi. (Betancourt, 2017)



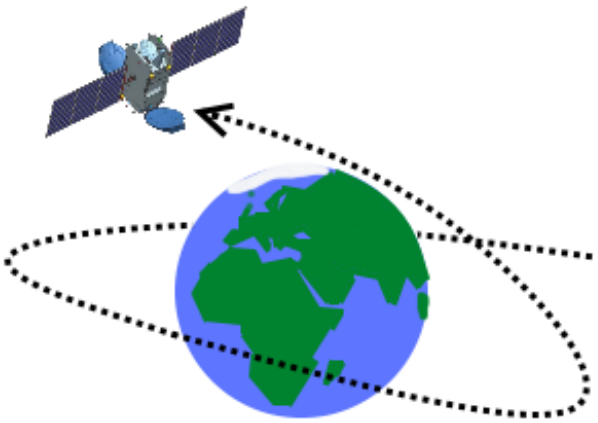
Kuvio 3: Hahmotelma jakauman dimension kasvattamisen vaikutuksesta. Musta ympyrä kuvaa jakaumien moodeja ja punainen alue kuvaa ns. tyypillistä aluetta (eng. *typical set*), joka vaikuttaa eniten odotusarvon suuruuteen. Vasemmalla on kaksiulotteinen jakauma ja oikealla 10-ulotteinen jakauma.

2.5 Hamiltonin Monte Carlo

Hamiltonin Monte Carlo perustuu vahvasti Metropolis-algoritmiin. Näiden molempien algoritmien toiminta voidaan jakaa kahteen osaan: uuden ehdotuksen generoimiseen ja hyväksymis/hylkäämis -vaiheeseen. Hamiltonin Monte Carlon kohdalla uuden ehdotuksen generoimisvaihe on kuitenkin huomattavasti monimutkaisempi. Metropolis-algoritmissa kohdejakauman sijainnista avaruudessa ei ole minkäänlaista tietoa, vaan uusi ehdotus tulee aina nykyisen pisteen lähiympäristöstä satunnaisesti. Hamiltonin Monte Carlo puolestaan pyrkii ratkaisemaan tyhjän tilan osuuteen liittyvän ongelman: sen sijaan, että uusi ehdotus valittaisiin täysin satunnaisesti sen hetkisen pisteen ympäristöstä, käytetään uuden ehdotuksen generoimisessa hyödyksi tietoa jakauman sijainnista. Tätä suuntaa voidaan ajatella vektorikenttänä, joka tulee kohdejakauman differentiaalirakenteesta.

2.5.1 Toimintaperiaate

Ilmiötä voidaan verrata maata kiertävään satelliittiin, missä maan painovoima vastaa jakauman differentiaalirakennetta (kuvio 4). Painovoiman vektorikenttä antaa jatkuvasti satelliitille tietoja maapallon sijainnista ja satelliitti pysyy maapallon läheisyydessä painovoiman ja liikemäärän avulla. Vastaavasti HMC-algoritmissa on sijaintiparametri, joka kertoo missä pisteessä algoritmi tällä hetkellä on sekä liikemääräparametri, joka kertoo mihin suuntaan seuraavan askeleen pitäisi mennä, jotta algoritmi pysyisi jakauman lähellä ja uudet ehdotukset Markovin ketjun arvoiksi olisivat mahdollisimman hyviä.



Kuvio 4: HMC-algoritmin etenemistä voidaan verrata satelliitin liikkeeseen maapallon ympäri. Satelliitin sijainti avaruudessa vastaa parametria θ, satelliitin nopeus ja suunta vastaavat liikemäärämuuttujaa ϕ ja painovoima vastaa posteriorijakauman differentiaalirakennetta.

Mallin parametrit θ_j vastaavat HMC:ssä sijaintimuuttujaa. Jokaiselle parametrille θ_j annetaan oma liikemäärämuuttujansa ϕ_j (eng. *momentum*), joka vaikuttaa uuden ehdotuksen generoimiseen. Tämän seurauksena parametriavaruus laajenee D-ulotteisesta $2D$-ulotteiseksi avaruudeksi, jota kutsutaan vaiheavaruudeksi (eng. *phase space*). Algoritmissa simuloidaan yhteisjakaumasta

$$p(\theta, \phi|y) = p(\phi)p(\theta|y). \tag{1}$$

Posteriorin laskennassa ollaan kiinnostuneita pääasiassa satunnaismuuttujasta θ ja ϕ on

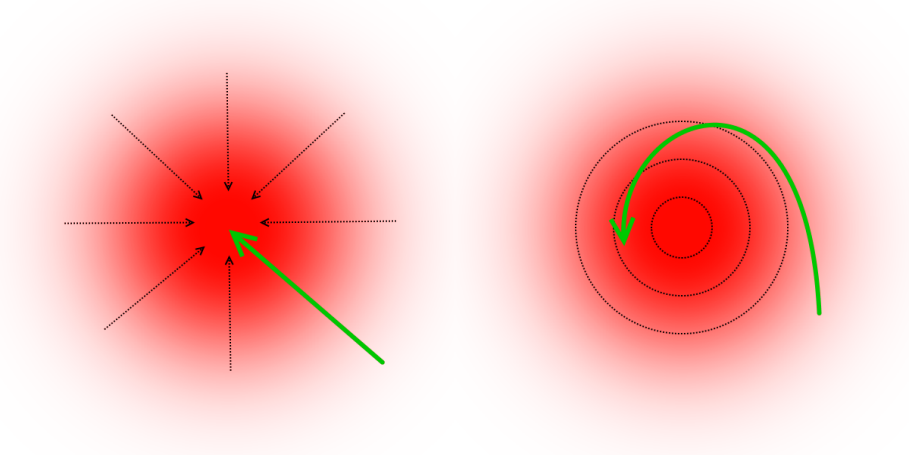
ainoastaan apumuuttuja, joka mahdollistaa algoritmin nopeamman liikkumisen parametriavaruudessa. Kun yhteisjakaumasta marginalisoidaan pois ϕ, alkuperäinen kohdejakauma saadaan palautettua.

Muuttuja ϕ kertoo differentiaalirakenteen avulla, missä suunnassa kiinnostavan posteriorijakauman moodi sijaitsee. Tästä syystä muuttujaa ϕ varten täytyy laskea log-posteriorin gradientti. Liikemäärämuuttujan ϕ arvoja ei algoritmin toimimiseksi kannalta tarvitse pitää muistissa, sillä ϕ arvotaan satunnaisesti jokaisen iteraation alussa. (Gelman et al., 2014, s. 301)

HMC:tä varten täytyy laskea log-posteriorin gradientti (vektoriderivaatta). Tyypillisesti tämä tehdään analyyttisesti, sillä numeerinen laskeminen on aikaavievää. Kun θ on d-dimensioinen, niin gradientti on

$$\frac{d\log(p(\theta|y))}{d\theta} = (\frac{d\log(p(\theta|y))}{d\theta_1}, ..., \frac{d\log(p(\theta|y))}{d\theta_d}).$$

Pelkkä gradientin suunnan noudattaminen ei tuota haluttua jakaumaa, vaan gradientti vie simulaation suoraan jakauman moodiin (kuvio 5). Vastaava tilanne painovoimaesimerkissä olisi satelliitin vieminen avaruuteen ilman liikemäärää: satelliitti ei jäisi maata kiertävälle kiertoradalle, vaan rysähtäisi suoraan maahan. Tästä johtuen algoritmissa liikemäärämuuttujaa ϕ päivitetään puoliaskelilla. (Betancourt, 2017)



Kuvio 5: Vasemmalla on kuvattu simulointia, jossa seuraava ehdotus otetaan suoraan gradientin avulla. Simuloinnissa lähestytään kohdejakauman moodia, mutta simulointi ei tuota kohdejakauman tyypillisiä arvoja. Oikealla simuloidut arvot vastaavat tyypillistä kohdejakauman otosta.

Oleellisin osa Hamiltonin Monte Carloa on uuden ehdotuksen generoiminen liikemäärämuuttujan ϕ avulla. Uusi ehdotus generoidaan liikkumalla ensin parametriavaruudessa päivittäen sijaintia ja liikemäärää sen mukaan, missä kohtaa parametriavaruutta algoritmi kulkee. Menetelmän ideaa voidaan verrata painovoimaesimerkkiin. Alkutilanteessa satelliitilla on jokin liikemäärä. Yhden aikayksikön jälkeen satelliitti on liikkunut painovoiman vaikutuksesta eli sen sijainti on päivittynyt liikemäärän avulla. Tämän jälkeen päivitetään satelliitin liikemäärävektori tässä uudessa sijainnissa. Kun tätä toistetaan usean aikayksikön verran, tulokseksi saadaan satelliitin liikerata maapallon ympäri. Vastaavasti HMC:ssä ehdotuksen generoimisen aikana käydään läpi pisteen liikerata parametriavaruudessa. Sen jälkeen kun piste on kulkenut halutun pituisen matkan, sen sijainti asetetaan uudeksi ehdotukseksi algoritmiin. Koska liikerata kulkee korkean todennäköisyyden alueella, uuden ehdotuksen hyväksymistodennäköisyys on suuri. Toisaalta uusi sijainti on kaukana vanhasta, jolloin Markovin ketjun arvojen välinen autokorrelaatio on pieni. (Marwala, 2012)

2.5.2 Algoritmin rakenne

HMC-algoritmi perustuu Hamiltonin mekaniikkaan, joka on eräs lähestymistapa klassiseen mekaniikkaan. HMC-algoritmissa yhteisjakaumaa $p(\theta, \phi)$ voidaan kuvata Hamiltonin funktion $H(\theta, \phi)$ avulla:

$$p(\theta, \phi) = e^{-H(\theta,\phi)}.$$

Tästä saadaan kaavan 1 nojalla:

$$H(\theta, \phi) = -\log(p(\theta, \phi)) = -\log(p(\phi|\theta)) - \log(p(\theta)),$$

missä termiä $-\log(p(\phi|\theta))$ kutsutaan kineettiseksi energiaksi $K(\phi, \theta)$ ja termiä $-\log(p(\theta))$ potentiaalienergiaksi $V(\theta)$. Hamiltonin funktio kuvaa tyypillisen alueen geometriaa ja Hamiltonin yhtälöiden avulla saadaan HMC-algoritmia varten haluttu vektorikenttä:

$$\frac{d\theta}{dt} = \frac{\partial H}{\partial \phi} = \frac{\partial K}{\partial \phi} \tag{2}$$

$$\frac{d\phi}{dt} = -\frac{\partial H}{\partial \theta} = -\frac{\partial K}{\partial \theta} - \frac{\partial V}{\partial \theta}, \tag{3}$$

missä $\frac{\partial V}{\partial \theta}$ on log-posteriorin gradientti. Hamiltonin yhtälöt siis kertovat, kuinka θ ja ϕ muuttuvat ajan t suhteen. Käytännössä Hamiltonin yhtälöitä approksimoidaan ns. pukkihyppy-vaiheessa (eng. *leapfrog*) äärellisellä askelten määrällä.

HMC:ssä algoritmin sijaintia ja liikemäärää päivitetään pukkihyppy-vaiheella, joka approksimoi Hamiltonin yhtälöitä (2) ja (3). Pukkihyppy-vaihe kuuluu ns. symplektisiin integrointimenetelmiin (eng. *symplectic integrators*), jotka ovat osoittautuneet erittäin tehokkaiksi menetelmiksi tarkasteltaessa Hamiltonin yhtälöitä (Leimkuhler & Reich, 2004, Hairer et al., 2006). Pukkihyppy-vaiheessa muodostetaan liikerata, jonka viimeinen piste on HMC-algoritmin uusi ehdotus. Menetelmä approksimoi hyvin tarkasti algoritmin liikerataa, mutta pienistä virheistä johtuen tulokset ovat harhaisia. Pukkihyppy-vaihe simuloi liikerataa seuraavasti:

Vaiheet 1-3 toistetaan L kertaa.

1. Liikemäärävektorin ϕ puoliaskel:
$$\phi \leftarrow \phi + \frac{1}{2} c \frac{d \log(p(\theta|y))}{d\theta}.$$
2. Liikemäärävektorin ϕ avulla päivitetään muuttujaa θ (sijainti):
$$\theta \leftarrow \theta + c\phi.$$
3. Liikemäärävektorin ϕ puoliaskel:
$$\phi \leftarrow \phi + \frac{1}{2} c \frac{d \log(p(\theta|y))}{d\theta}.$$

Pukkihyppyvaiheessa c on käyttäjän syöttämä vakio. Varsinaisessa HMC-algoritmissa vakio c on korvattu hienosäätöparametreilla ϵ ja M. Tämä muutos on tehty käytännöllisyyssyistä: algoritmin toimintaa voidaan tarkastella eri parametrin ϵ arvoilla, kun M pidetään kiinnitettynä. (Gelman et al., 2014, s. 301)

Tuloksien harhaisuus voidaan korjata impletoimalla Metropolis-Hastings-algoritmin mukainen hyväksymis/hylkäämisvaihe Hamiltonin siirtymän pisteelle vaiheavaruudessa. Tämä varmistaa sen, että simuloitu otos on posteriorijakaumasta. Ehdotuksen hyväksymistodennäköisyyden α laskeminen yksinkertaistuu tässä tapauksessa Metropolis-algoritmin vastaavaksi todennäköisyydeksi, mutta pelkän parametriavaruuden sijaan tarkastellaankin vaiheavaruutta. Todennäköisyyden α konstruoinnissa tulee ottaa huomioon HMC-algoritmin ominaisuudet. (Betancourt, 2017)

Todennäköisyyden α laskemisessa ei voida käyttää suoraan samaa ideaa kuin Metropolis-algoritmissa: liikerata ei ole luonnostaan kääntyvä ja kun tarkastellaan todennäköisyyttä siirtyä tilasta (θ_L, ϕ_L) ajassa taaksepäin tilaan (θ_0, ϕ_0), niin tämä todennäköisyys $q(\theta_L, \phi_L|\theta_0, \phi_0) = 0$, jolloin suhde

$$\frac{q(\theta_L, \phi_L|\theta_0, \phi_0)}{q(\theta_0, \phi_0|\theta_L, \phi_L)} = \frac{1}{0}.$$

Tämä ongelma voidaan korjata yksinkertaisesti vaihtamalla liikemäärän ϕ merkkiä:

$$(\theta, \phi) \to (\theta, -\phi),$$

jolloin ehdotuksen hyväksymistodennäköisyys

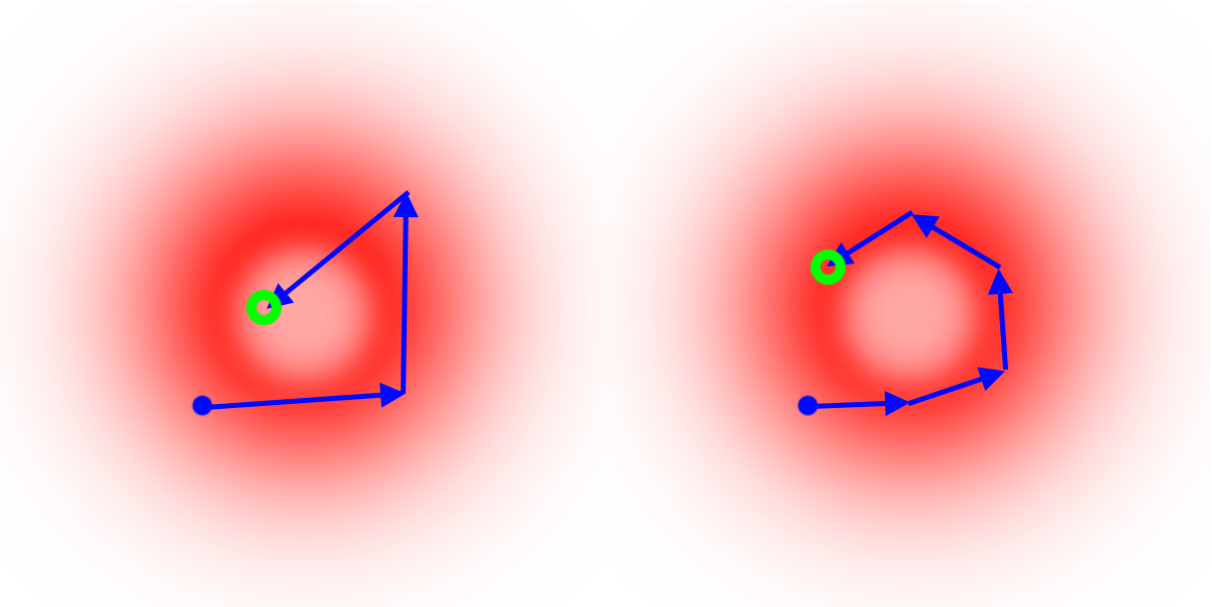
$$\begin{aligned}
\alpha_{hmc} &= \min\Big(1, \frac{p(\theta_L, -\phi_L|y)q(\theta_L, -\phi_L|\theta_0, \phi_0)}{p(\theta_0, \phi_0|y)q(\theta_0, \phi_0|\theta_L, -\phi_L)}\Big) \\
&= \min\Big(1, \frac{p(\theta_L, -\phi_L|y)}{p(\theta_0, \phi_0|y)}\Big) \\
&= \min\Big(1, \frac{\exp(-H(\theta_L, -\phi_L))}{\exp(-H(\theta_0, \phi_0))}\Big) \\
&= \min(1, \exp(H(\theta_L, -\phi_L) - H(\theta_0, \phi_0)).
\end{aligned}$$

Tämän seurauksena Markovin ketju on kääntyvä ja ketjun tasapainojakauma vastaa posteriorijakaumaa.

Tavallisesti apumuuttujan ϕ ajatellaan noudattavan multinormaalijakaumaa odotusarvolla 0 ja kovarianssimatriisilla M (eng. *mass matrix*). Matriisi M asetetaan normaalisti diagonaalimatriisiksi, jolloin muuttujat ϕ ovat riippumattomia. Algoritmi toimii millä tahansa diagonaalimatriisilla, mutta algoritmi on tehokkaampi, kun matriisi M on skaalattu noin posteriorin käänteisen kovarianssimatriisin $(var(\theta|y))^{-1}$ suuruiseksi. (Gelman et al., 2014, s. 301)

Ehdotuksen generoimiseen käytetyn liikeradan pituus riippuu yhden askeleen pituudesta ϵ ja askelten määrästä L. Tavallisesti niiden arvot asetetaan siten, että $\epsilon L = 1$, jolloin liikemäärän ϕ asteikkoparametrit vastaavat kohdejakauman asteikkoa. Yhtälön $\epsilon L = 1$ seurauksena algoritmi etenee pukkihyppy-vaiheen aikana suurin piirtein jakauman toiselle puolelle. Tällöin myös Markovin ketjun arvojen autokorrelaatio on pientä, sillä peräkkäiset arvot ovat kaukana toisistaan. Näitä ns. hienosäätöparametreja muuttamalla voidaan vaikuttaa algoritmin toimintaan. Lyhyillä askelilla ϵ algoritmin generoimat ehdotukset pysyvät lähellä jakauman tyypillisiä arvoja ja ehdotuksia hyväksytään usein.

Tällöin kuitenkin kasvatetaan askelten määrää, jolloin gradientti lasketaan useamman kerran yhden pukkihyppy-vaiheen aikana ja tämä lisää laskenta-aikaa. Pitkillä askelilla ja lyhyellä askelmäärällä taas hyväksymistodennäköisyys on pienempi ja iteraatioden määrää pitää lisätä (kuvio 6). Optimaalinen askelpituus ja askelmäärä riippuvat tilanteesta, mutta hyvänä tavoitteena voidaan pitää 65 %:n hyväksymistodennäköisyyttä. (Gelman et al., 2014, s. 301)



Kuvio 6: Kuviossa on hahmoteltu hienosäätöparametrien vaikutus uuden ehdotuksen generoimiseen. Vasemmalla askeleet ovat suurempia, jolloin askelien määrä on pienempi. Tällöin uuden ehdotuksen generoimisessa pukkihyppy-vaihe käydään läpi kolme kertaa. Oikealla taas askeleet ovat pienempiä, jolloin askelten määrä on suurempi kuin vasemmalla ja pukkihyppy-vaihe käydään läpi viisi kertaa.

Oikeastaan HMC-algoritmi koostuukin siis kolmesta askeleesta: pukkihyppy-vaiheesta, hyväksymistodennäköisyyden laskemisesta ja ehdotuksen hyväksymis/hylkäämis -vaiheesta (Gelman et al., 2014, s. 301-302).

1. Alustetaan ϕ arpomalla satunnainen arvo normaalijakaumasta (odotusarvo 0, kovarianssi M). Seuraavaksi päivitetään muuttujia θ ja ϕ yhtä aikaa L kertaa, skaalattuna kertoimella ϵ. Pukkihyppy-vaihe a-c käydään läpi L kertaa:

   (a) Liikemäärävektorin ϕ puoliaskel:

   $$\phi \leftarrow \phi + \frac{1}{2}\epsilon \frac{d \log(p(\theta|y))}{d\theta}.$$

   (b) Liikemäärävektorin ϕ avulla päivitetään muuttujaa θ (sijainti):

   $$\theta \leftarrow \theta + \epsilon M^{-1}\phi.$$

   (c) Liikemäärävektorin ϕ puoliaskel:

   $$\phi \leftarrow \phi + \frac{1}{2}\epsilon \frac{d \log(p(\theta|y))}{d\theta}.$$

2. Lasketaan

   $$r = \frac{p(\theta^*|y)p(\phi^*)}{p(\theta^{t-1}|y)p(\phi^{t-1})},$$

   missä θ^{t-1} ja ϕ^{t-1} ovat arvot ennen pukkihyppy-prosessia ja θ^* ja ϕ^* ovat arvot prosessin jälkeen

3. Todennäköisyydellä $\min(r, 1)$ asetataan $\theta^t = \theta^*$, muulloin $\theta^t = \theta^{t-1}$. Liikemäärämuuttujan ϕ arvoa ei tarvitse säilyttää, sillä sen arvo päivitetään seuraavan iteraation alussa.

2.5.3 Haasteet ja aiempi tutkimus

Hamiltonin Monte Carlo -menetelmän implementointi erilaisiin malleihin voi olla haastavaa jopa kokeneelle ammattilaiselle (Ishwaran, 1999, Neal, 2010). Algoritmia varten tulee laskea logaritmisoidun posteriorin gradientti ja tämä voi olla hankalaa ja aikaavievää monimutkaisten mallien tapauksissa. Ennen algoritmin käyttöönottoa tulisikin tarkistaa

mahdolliset huolimattomuusvirheet vertailemalla analyyttisesti laskettua gradienttia numeerisesti laskettuun. Myös hienosäätöparametrien valinta voi tuottaa hankaluuksia, sillä HMC:n tehokkuus on täysin riippuvainen valittujen parametrien arvoista. Huonosti valitut ϵ, L tai matriisi M voivat lisätä simulointiin käytettyä aikaa, parantamatta kuitenkaan algoritmin toimintaa. Tyypillisesti HMC on ergodinen (eng. *ergodic*), eli algoritmin muodostama liikerata ei jää jumiin tiettyyn osaan parametriavaruutta. Tietyillä parametrien ϵ ja L arvoilla tämä ei kuitenkaan päde: kun tulo ϵL on noin 2π, niin pukkihyppy-vaiheessa palataan takaisin aloituspisteeseen, missä se oli ennen pukkihyppy-vaihetta (Neal, 2010). Neal (2010) toteaa myös, että hienosäätöparametrien asettaminen "yrityksen ja erehdyksen kautta"on usein välttämätöntä.

Myös monihuippuiset jakaumat voivat tuottaa hankaluuksia HMC:lle: jos posteriorijakauma on kaksihuippuinen ja moodit ovat kaukana toisistaan, pukkihypyn muodostama liikerata ei koskaan löydä koko jakaumaa (Neal, 2011). Tämä johtuu siitä, että algoritmilla ei missään vaiheessa ole tarpeeksi liikemäärää päästäkseen tyhjän tilan yli toisen moodin lähelle. Kuten Ip & Jewson (2010) osoittivat, HMC:ssä voi olla tilanteita, joissa Markovin ketju saa arvoja vain toisen moodin läheltä. Metropolis-algoritmi sen sijaan usein löytää molemmat moodit, mutta konvergoiminen vaatii useita iteraatioita. Kun Metropolis-algoritmin ehdotusjakauma on tarpeeksi leveä, se voi hypätä tyhjän tilan yli toiselle moodille (Ip & Jewson, 2010).

Useat tutkijat ovat verranneet Hamiltonin Monte Carloa muihin algoritmeihin (Neal, 2010, Nugraho & Morimoto, 2015, Ip & Jewson, 2010). Neal (2010) vertaa julkaisussaan HMC:tä Metropolis-algoritmiin tilanteissa, joissa posteriorijakaumana toimii multinormaalijakauma. HMC osoittautui tehokkaammaksi menetelmäksi tilanteissa, joissa parametrien välinen korrelaatio on suurta (noin 0,98) sekä tilanteissa, joissa tarkastellaan 100-ulotteista multinormaalijakaumaa (Neal, 2010). Myös Ip & Jewson (2010) tarkastelevat raportissaan HMC:n ja Metropolis-algoritmin toimintaa multinormaalijakaumissa: he tutkivat, miten hyväksymistodennäköisyys muuttuu, kun jakauman dimensiota kasvatetaan. Metropolis-algoritmissa hyväksymistodennäköisyys putoaa alle 10 prosenttiin, kun dimensio on suurempi kuin 10. HMC:ssä se puolestaan pysyy lähellä 100 prosenttia myös 50 parametrin jakaumassa. Nugraho & Morimoto (2015) puolestaan vertailevat työssään Hamiltonin Monte Carloa Riemann Manifold HMC -menetelmään ja multi-move Met-

ropolis Hastings -menetelmään (MM-MH). Artikkelissa menetelmiä sovelletaan erilaisiin stokastisen volatiliteetin malleihin, joissa mallinnetaan volatiliteettia finanssiaikasarjoissa ja tutkimus osoittaa Hamiltonin Monte Carloon perustuvien algoritmien olevan tehokkaampia kuin MM-MH (Nugraho & Morimoto, 2015).

2.5.4 Variaatioita

HMC-algoritmista on kehitelty useita menetelmään perustuvia laajennoksia, joille yhteistä on ehdotuksen generoiminen Hamiltonin yhtälöiden avulla. Näiden laajennosten tavoitteena on toisaalta parantaa algoritmin toimintaa monimutkaisissa tilanteissa, toisaalta automatisoida uuden ehdotuksen generoimista siten, että algoritmin tehokkuus ei riippuisi käyttäjän syöttämistä hienosäätöparametrien arvoista. Useat HMC:n laajennokset muuttavatkin hienosäätöparametrien arvoja sitä mukaan, kun algoritmi etenee parametriavaruudessa. Esimerkiksi parametri ϵ voidaan asettaa pienemmäksi, kun pukkihyppy-vaiheessa ollaan alueella, jossa jakauman kaareutuvuus on suurta (Gelman et al., 2014, s. 304). Ajon aikana hienosäätöparametria L voidaan myös muuttaa siten, että uusi ehdotus olisi mahdollisimman kaukana nykyisestä pisteestä, mutta algoritmi ei käyttäisi laskenta-aikaa tarpeettoman pitkään liikerataan. (Gelman et al., 2014, s. 304-305)

Yksi esimerkki tällaisesta laajennoksesta on No-U-turn sampler (NUTS). Siinä parametrin L arvo optimoidaan automaattisesti jokaisen iteraation aikana. Nimi No-U-turn viittaa siihen, että pukkihyppy-vaihe päättyy ennen kuin liikerata ehtii kääntymään kohti aloituspistettä. Tämä toteutetaan optimoimalla parametri L jokaisen kierroksen alussa. Tällöin generoidut ehdotukset ovat kaukana toisistaan, eikä algoritmi käytä turhaa laskentatehoa liian pitkiin liikeratoihin. Menetelmä on huomattavasti monimutkaisempi kuin tavallinen HMC mutta yhtä tehokas tai jossain tilanteissa jopa sitä tehokkaampi. (Hoffman & Gelman, 2014)

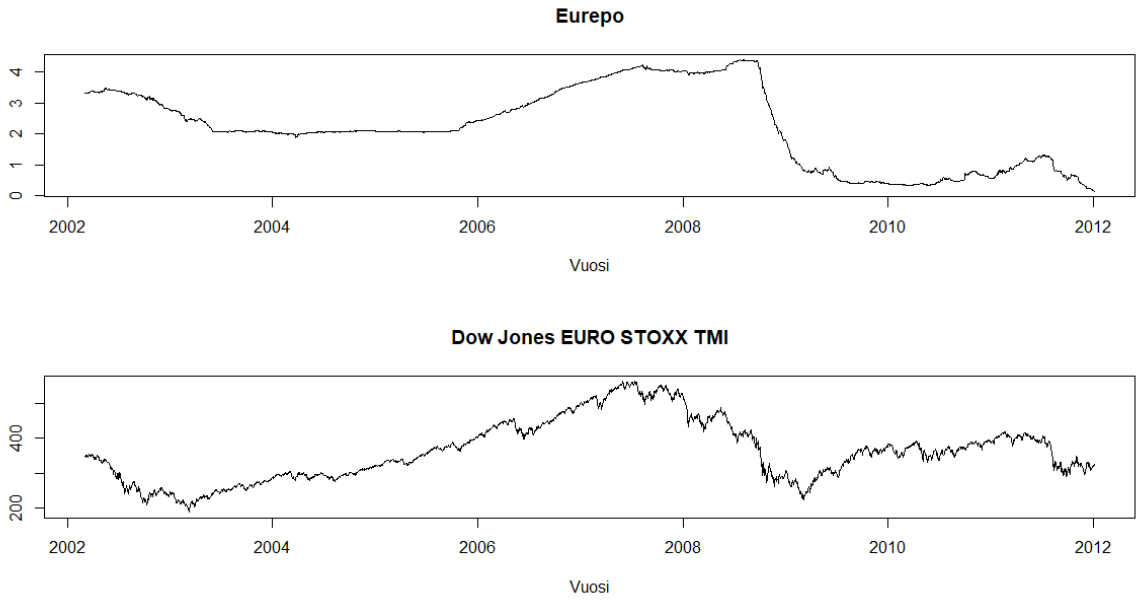
Toinen suosiota kerännyt HMC-algoritmin laajennos on Riemann Manifold HMC (RMHMC), jossa matriisi M sopeutetaan posteriorin paikalliseen kaareutuvuuteen (eng. *local curvature*) jokaisen iteraation aikana. Myös RMHMC on perinteistä HMC:tä monimutkaisempi ja tehokkaampi. (Girolami & Calderhead, 2011)

3 Sovelluskohde

Tässä työssä Hamiltonin Monte Carloa sovelletaan esimerkinomaisesti kahteen finanssiaikasarjamalliin. Tavoitteena on valita realistisia ja monimutkaisia monen muuttujan malleja ja selvittää, mitä tulee ottaa huomioon sovellettaessa HMC:tä tämän tyyppisiin malleihin. Toisena tavoitteena on vertailla HMC:tä ja Metropolis-algoritmia toisiinsa mm. tehokkuuden, käytännöllisyyden ja autokorrelaation suhteen. Tavoitteiden kannalta itse Bayes-analyysin tulokset eivät ole oleellisia, vaan ainoastaan algoritmien toiminta kyseisissä malleissa. Tässä luvussa esitellään lyhyesti työssä käytetty aineisto ja kuvaillaan kahden aikasarjamallin uskottavuusfunktiot ja priorit.

3.1 Aineisto

Mallin aineistona käytetään kaksiulotteista sarjaa, jossa osakeindeksisarjana toimii Dow Jones Euro Stoxx -indeksi ja korkosarjana Eurepo (kuvio 7). Molemmat aikasarjat ovat vuosilta 2002-2011 ja havaintojen lukumäärä on 2510. Indeksi- ja korkosarja ovat verkossa vapaasti saatavilla: tämän työn aineisto on otettu Tampereen yliopiston palvelimelta osoitteesta http://www.sis.uta.fi/tilasto/codes/savings/index-interest-data.csv.



Kuvio 7: Korkosarja ja osakeindeksisarja

Tämän työn malleissa mallinnetaan säästöhenkivakuutuksia. Säästöhenkivakuutus, joka tunnetaan myös nimellä sijoitusvakuutus, ei periaatteessa ole vakuutus, vaan ainoas-

taan tapa sijoittaa. Vakuutettu maksaa vakuutusyhtiölle sopimuskauden alussa sovitun summan ja vakuutusyhtiö sijoittaa nämä maksut haluamallaan tavalla mahdollisimman hyvin. Sopimuskauden lopussa summa maksetaan takaisin asiakkaalle korkoineen. Koron suuruus riippuu vakuutuksen tyypistä. Sijoitussidonnaisessa vakuutuksessa tuotto maksetaan sijoitusten arvon kehityksen mukaan, kun taas laskuperustekorkoisessa vakuutuksessa tuotto muodostuu ennalta sovitusta kiinteästä korosta tai johonkin korkotekijään sidotusta korosta, kuten kolmen kuukauden euriborista. Jälkimmäisessä tyypissä asiakas voi saada kiinteän koron lisäksi bonuskorkoa (eng. *bonus rate*), jos vakuutusyhtiön sijoitukset ovat olleet erityisen tuottoisia. (Finanssivalvonta 2017)

Säästöhenkivakuutukset ovat usein suuri riski vakuutusyhtiöille niiden sisältämien välillisten optioiden vuoksi (eng. *implicit options*). Nämä optiot ovat vakuutuksen erilaisia ominaisuuksia, kuten kiinteä korkotaso ja vakuutetun oikeus muokata vakuutussopimusta kesken sopimuskauden. Mahdollisia uhkia ovat mm. tuotteiden väärinhinnoittelu ja puutteellinen riskien hallinnointi (Gatzert, 2009), jotka olivat osaltaan syy The Equitable Life Assurance Society -yhtiön taloudelliseen kriisiin vuonna 2000 (O'Brien, 2006). Tämän tapahtuman jälkeen huoli näiden optioiden riskeistä on kasvanut. Euroopan unionin Solvency II -direktiivin tarkoitus onkin kannustaa vakuutusyhtiöitä riskien mittaamiseen ja hallitsemiseen mallien avulla. Direktiivi tuli voimaan vuoden 2016 tammikuussa.

3.2 Finanssimallit

Työssä tavoitteena on sovittaa HMC monimutkaiseen, moniulotteiseen ja ennen kaikkea realistiseen malliin. Finanssiaikasarjamallit ovat moniulotteisia ja niiden monimutkaisuutta voidaan kasvattaa lisäämällä malliin erilaisia prosesseja. Bayes-menetelmien suosio finanssimalleissa on kasvanut räjähdysmäisesti Bayes-vallankumouksen jälkeen: MCMC-menetelmien avulla voidaan tarkastella entistä moniulotteisimpia malleja ja parametrien epävarmuuden kuvailu on tarkkaa ja selkeää. Tämän työn tavoitteena on välttää keinotekoisia malleja, kuten multinormaalijakaumaa vahvasti riippuvilla parametreilla (esim. Ip & Jewson (2010) ja Neal (2010)), ja sen sijaan tarkastella malleja, jotka ovat hyvin tavallisia tutkimuksissa. Lisätietoa finanssimallinnuksesta löytyy teoksesta Investment Guarantees (Hardy, 2003).

Tässä työssä sovelletaan sekä Hamiltonin Monte Carlo -menetelmää että Metropolis-

algoritmia kahteen Luoman ja Puustellin (Luoma & Puustelli, 2009) esittämään finanssimalliin, joissa korkotaso ja volatiliteetti ovat stokastisia. Mallien parametrien arvot ovat tuntemattomia, joten bayesiläinen lähestymistapa on luonteva tapa arvioida parametrien epävarmuutta. Molempien mallien avulla mallinnetaan osake- ja korkosijoitusten tuottoa.

3.2.1 Malli A

Malli A (Luoma & Puustelli, 2009) on hyvin tavallinen ja realistinen kahdeksan parametrin malli, joka kuvaa varallisuutta ja korkotasoa. Olkoon $Z_t^{(i)}, i = 1, 2, 3$ standardeja Brownin liikkeitä. Oletetaan, että $Z_t^{(1)}$ ja $Z_t^{(3)}$ ovat riippumattomia ja $Z_t^{(1)}$ ja $Z_t^{(2)}$ välinen korrelaatiorakenne on (Luoma et al., 2013):

$$Z_t^{(2)} = \rho Z_t^{(1)} + \sqrt{1+\rho^2} Z_t^{(3)}.$$

Oletetaan, että riskitön korkotaso prosentteina x_t noudattaa stokastista differentiaaliyhtälöä

$$dx_t = (\beta - \kappa x_t)dt + \tau {x_t}^{\gamma} d{Z_t}^{(1)},$$

ja osakeindeksi noudattaa varianssin vakiojousto-prosessia

$$dS_t = r_t S_t dt + \nu S_t^{1-\alpha} dZ_t^{(2)}.$$

Simuloinnin kannalta oleellinen uskottavuusfunktio $p(y|\theta)$ voidaan kirjoittaa muodossa:

$$\begin{aligned} p(y|\theta) = &\prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\tau^2 x_{(i-1)\delta}^{2\gamma}\delta}} \exp\left(- \frac{(x_{i\delta} - x_{i-1\delta} - (\beta - \kappa x_{(i-1)\delta})\delta)^2}{2\tau^2 x_{(i-1)\delta}^{2\gamma}\delta} \right) \\ &\times \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\nu^2 S_{(i-1)\delta}^{2(1-\alpha)}(1-\rho^2)\delta}} \exp\left(- \frac{(S_{i\delta} - S_{i-1\delta} - \mu S_{(i-1)\delta}\delta - \nu S_{(i-1)\delta}^{1-\alpha}\rho\Delta Z_{i\delta}^{(1)})^2}{2\nu^2 S_{(i-1)\delta}^{2(1-\alpha)}(1-\rho^2)\delta} \right), \end{aligned} \tag{4}$$

missä $y = (x, S)$ on data, $\theta = (\mu, \nu, \alpha, \beta, \kappa, \tau, \gamma, \rho)$ ja $\Delta Z_{i\delta}^{(1)} = \frac{x_{i\delta} - x_{(i-1)\delta} - (\beta - \kappa x_{(i-1)\delta})\delta}{\tau x_{(i-1)\delta}^{\gamma}}$.
Prioriksi valitaan epäaito priori:

$$p(\theta) = \begin{cases} 1, & \text{kun } |\rho| < 1 \text{ ja } \min(\kappa, \beta, \tau, \alpha) > 0, \\ 0, & \text{muuten.} \end{cases}$$

Parametrille τ^2 tehdään logaritmimuunnos.

3.2.2 Malli B

Malli B on muuten sama kuin malli A, mutta osakeindeksin volatiliteetissa käytetään hyödyksi tavallisen GARCH-mallin laajennosta (Engle & Ng, 1993) tehden mallista astetta monimutkaisemman kuin malli A. Mallissa B on yhteensä 11 parametria. Osakeindeksin logaritmin yhtälö on

$$\log(S_t/S_{t-1}) = \alpha + \lambda\sqrt{h_t} - \frac{1}{2}h_t + \sqrt{h_t}z_t,$$

missä $z_t \sim N(0,1)$. Prosessit h_t ja z_t päivitetään kaavoilla:

$$h_t = \beta_0 + \beta_1 h_{t-1} + \beta_2 h_{t-1}(z_{t-1} - \mu)^2$$

ja

$$z_t = \frac{\Delta\log(S_{t-1}) - (\alpha + \lambda\sqrt{h_t} - 0.5h_t)}{\sqrt{h_t}}$$

Korkotaso prosentteina x_t noudattaa seuraavaa prosessia:

$$dx_t = (\beta - \kappa x_t)dt + \tau {x_t}^{\gamma} d{Z_t}^{(1)}.$$

Mallin B uskottavuusfunktio on tällöin

$$\begin{aligned} p(y|\theta) = &\prod_{t=1}^{N} \frac{1}{\sqrt{2\pi\tau^2 x_{(t-1)}^{2\gamma}}} \exp\left(- \frac{(x_t - x_{t-1} - (\beta - \kappa x_{(t-1)}))^2}{2\tau^2 x_{(t-1)}^{2\gamma}} \right) \\ &\times \prod_{t=1}^{N} \frac{1}{\sqrt{2\pi h_t(1-\rho^2)}} \exp\left(- \frac{(\log(S_t/S_{t-1}) - \alpha - \lambda\sqrt{h_t} + 0.5h_t - \sqrt{h_t}\rho\Delta Z_t^{(1)})^2}{2h_t(1-\rho^2)} \right), \end{aligned} \tag{5}$$

missä $y = (x, S)$ on data, $\theta = (\kappa, \beta, \tau^2, \gamma, \rho, \mu, \beta_0, \beta_1, \beta_2, \alpha, \lambda)$ ja $\Delta Z_t^{(1)} = \frac{x_t - x_{(t-1)} - (\beta - \kappa x_{(t-1)})}{\tau x_{(t-1)}^{\gamma}}$.

Priori on sama kuin mallissa A:

$$p(\theta) = \begin{cases} 1, & \text{kun } |\rho| < 1 \text{ ja } \min(\kappa, \beta, \tau, \alpha) > 0, \\ 0, & \text{muuten.} \end{cases}$$

Parametrille τ^2 tehdään logaritmimuunnos.

4 Tulokset

Tässä luvussa sovelletaan sekä HMC, että Metropolis-algoritmeja edellisessä luvussa esitettyihin malleihin. Algoritmeja verrataan suoritusajan, autokorrelaation ja käytännöllisyyden suhteen.

4.1 Algoritmien toteutukset

Hamiltonin Monte Carlo ja Metropolis-algoritmi toteutetaan tässä tutkimuksessa R-ohjelmistolla (R Core Team, 2013). Molemmissa algoritmeissa on silmukkarakenteita ja tästä syystä R-kieli on laskennallisesti hitaampi kuin esimerkiksi erilaiset C-kielet. Varsinaisella nopeudella ei ole kuitenkaan merkitystä tämän työn kannalta, sillä tavoitteena on vertailla HMC:n ja Metropolis-algoritmin suoritusaikoja toisiinsa. Algoritmien implementointi R-kielen avulla tuo mukanaan myös muutamia etuja: erilaisten R-pakettien tarjoamat työkalut ovat hyödyllisiä mm. inferenssin tarkastelussa ja otospolkujen piirtämisessä, ja ennen kaikkea R on tuttu ohjelmointikieli monille tilastotieteen ammattilaisille. Mikäli tavoitteena olisi mahdollisimman nopea suoritusaika, tällöin suositellaan käytettäväksi C++-kieleen perustuvaa Stan-ohjelmaa, jota voi tarvittaessa käyttää R-ohjelmistosta käsin RStan-paketin (Stan development team, 2017) avulla. Stanin avulla voidaan soveltaa annettuun aineistoon ja malliin HMC- tai NUTS-algoritmia automaattisesti.

HMC:tä varten lasketaan posteriorin logaritmin gradientti. Gradientin laskeminen tehdään analyyttisesti, mutta mallien monimutkaisuuden takia tulokset on hyvä tarkistaa huolimattomuusvirheiden varalta. R-paketin numDeriv (Gilbert & Varadhan, 2016) grad-funktio laskee derivaatan numeerisesti ja tätä arvoa voidaan verrata analyyttisesti lasketuun arvoon jokaisen parametrin suhteen. Tässä työssä gradientti laskettiin muutamassa esimerkkipisteessä ja saatuja tuloksia verrattiin grad-funktion vastaaviin arvoihin. Arvot täsmäsivät viiden desimaalin tarkkuudella.

HMC-algoritmi on tehokkaimmillaan, kun hienosäätöparametrien M, L ja ϵ arvot on valittu mahdollisimman optimaalisesti. Nämä optimaaliset arvot ovat hyvin mallikohtaisia ja algoritmin toimintaa tuleekin testata monta kertaa useilla eri arvoilla. Kuten aiemmin mainittiin, matriisi M asetetaan posteriorijakauman käänteisen kovarianssimatriisin suuruiseksi. Koska parametrien suuruusluokka ei ole tunnettu, asetetaan $M = \text{Diag}(1, ..., 1)$. Kun simulointi on tehty näillä matriisin M arvoilla onnistuneesti, voidaan saatujen kes-

kivirheiden avulla muuttaa arvot tarkemmiksi. ϵ ja L vastaavat taas pukkihyppy-vaiheen yhden askeleen pituutta ja askelien määrää. Sen sijaan että asetetaan ϵ ja L vakioiksi, käytetään ajon aikana satunnaisesti muuttuvia arvoja. Tämä varmistaa, että algoritmi ei jää missään vaiheessa jumiin. Ennen jokaista pukkihyppy-vaihetta ϵ arvotaan väliltä $[\epsilon_0 - 0.2\epsilon_0, \epsilon_0 + 0.2\epsilon_0]$, missä ϵ_0 on ennen simulointia valittu keskimääräinen arvo muuttujalle ϵ. Vastaavasti L arvotaan väliltä $[L_0 - 0.4L_0, L_0 + 0.4L_0]$, missä L_0 vastaa keskimääräistä kierrosten määrää. Tämän lisäksi L pyöristetään lähimpään kokonaislukuun. Aluksi valitaan oletusarvoisesti $\epsilon_0 = 0.1$ ja $L_0 = 10$ ja näitä arvoja muutetaan hyväksymistodennäköisyyden ja ketjujen sekoittumisen mukaan.

Metropolis-algoritmin ehdotusjakaumana käytetään multinormaalijakaumaa varianssilla $(2.4/\sqrt{d})^2\Sigma$, missä d on mallin parametrien määrä ja Σ on jakauman kovarianssimatriisin approksimaatio. Matriisi Σ optimoidaan tässä työssä R-funktion nlm avulla: nlm-funktio palauttaa suurimman uskottavuuden estimaatin $\hat{\theta}$ ja matriisi Σ saadaan negatiivisen log-uskottavuusfunktion Hessen matriisin avulla.

HMC- ja Metropolis-algoritmia vertaillaan suoritusaikojen, autokorrelaation ja käytännöllisyyden suhteen. Jotta suoritusaikojen vertailu olisi järkevää, tulee varmistaa, että algoritmien tuloksissa on saavutettu riittävä tarkkuus ja että saadut tulokset ovat samankaltaisia. Pelkkä iteraatiokierrosten määrä ei ole hyvä mittari tälle tarkkuudelle, sillä MCMC-menetelmissä peräkkäiset arvot ovat usein vahvasti korreloituneita, jolloin tulokset ovat selvästi epätarkempia kuin vastaavat tulokset riippumattomista havainnoista. Ketjujen suppenemista voidaan arvioida potentiaalisen skaalan pienenemisen kertoimen (eng. *potential scale reduction factor*, PSRF) $\hat{R}$ avulla, kun taas tehokasta otoskokoa n_{eff} käytetään simuloinnin tarkkuuden arvioimiseen.

Kerroin $\hat{R}$ estimoi, kuinka paljon Markovin ketjun arvojen jakauman skaala voisi pienentyä, jos iteraatioiden määrä $n \to \infty$ (Gelman & Rubin, 1992). Kertoimen laskeminen perustuu usean ketjun simuloimiseen ja näiden ketjujen väliseen varianssiin B ja sisäiseen varianssiin W. Kerroin lasketaan yksittäisen parametrin posteriorivarianssin estimaatin $\hat{V}$ ja ketjun sisäisen varianssin W suhteella

$$\hat{R} = \sqrt{\frac{\hat{V}}{W}},$$

missä $\hat{R} \to 1$, kun $n \to \infty$ (Gelman et al., 2014, s. 284-285). Kun $\hat{R} < 1.1$, niin voidaan

päätellä, että suppeneminen on onnistunut (Brooks & Gelman, 1998).

Toinen keino vertailla algoritmeja on laskea tehokas otoskoko n_{eff}. Tehokas otoskoko kertoo, kuinka montaa riippumatonta havaintoa simuloitu otos vastaa. Se lasketaan kaavalla

$$n_{eff} = \frac{mn}{1 + 2\sum_{t=1}^{T} \hat{\rho}_t},$$

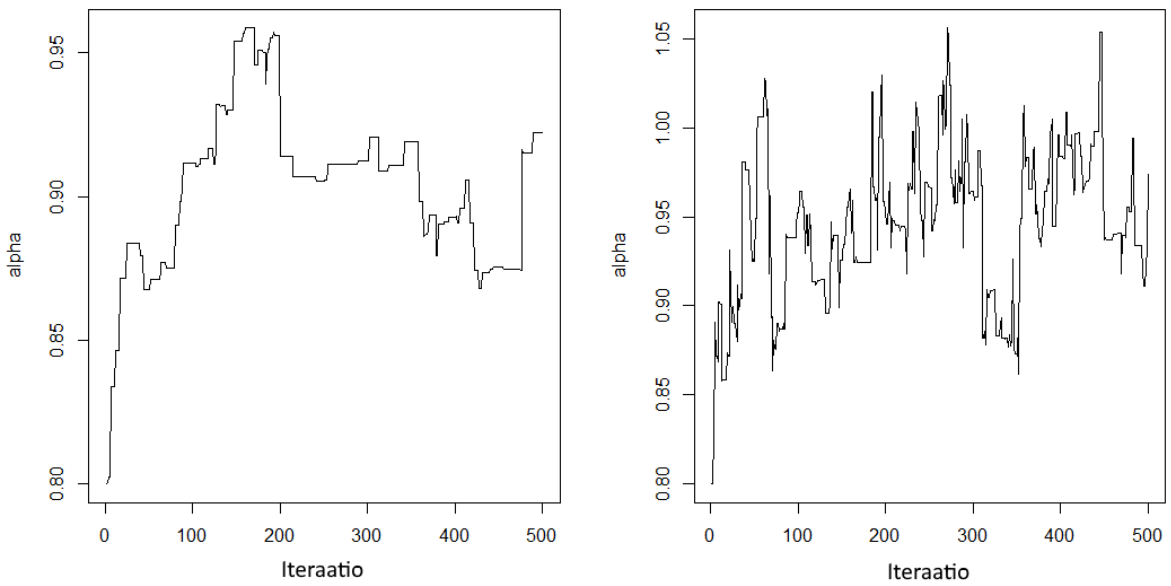
missä m on ketjujen määrä, n on iteraatioiden määrä ketjussa (burn-in vaihe poistettu) ja $\hat{\rho}_t$ on parametrin estimoitu autokorrelaatio viiveellä t. (Gelman et al., 2014, s. 286-287)

Algoritmien nopeutta voidaan vertailla laskemalla tehokkaan otoskoon suhde simulointiin käytettyyn aikaan parametreittain: $n_{eff,T} = n_{eff}/T_{sim}$. Kun T_{sim} on ilmoitettu sekunteina, kertoimen $n_{eff,T}$ avulla voidaan tarkastella, kuinka monta riippumatonta havaintoa algoritmi on tuottanut posteriorijakaumasta yhden sekunnin aikana.

4.2 Malli A

Ensimmäinen tavoite on saada onnistunut konvergoiminen HMC-algoritmin hienosäätöparametrien alkuarvoilla. Tällöin matriisi M voidaan asettaa oikeaan suuruusluokkaan, minkä jälkeen suoritetaan uusi optimaalinen simulaatio. Hienosäätöparametrien alkuarvoilla algoritmi etenee onnistuneesti mutta hitaasti ja uuden ehdotuksen hyväksymistodennäköisyys on liian pieni: n. 30 %. Hyväksymistodennäköisyyttä voidaan kasvattaa pienentämällä algoritmin askelia $\epsilon_0 = 0.01$, ja tämän seurauksena lisätään askelien määrää $L_0 = 100$ (kuvio 8), jolloin hyväksymistodennäköisyys on noin 65 %. Onnistuneen simuloinnin tuloksien avulla voidaan päätellä matriisin M arvojen suuruusluokka: rStan-paketin funktiolla *monitor* voidaan tarkastella parametrien posteriorihajontoja, joiden avulla asetetaan matriisin diagonaalin arvot (Stan development team, 2017). Keskihajontojen perusteella asetetaan $M = \text{Diag}(1/0.065^2, 1/0.333^2, 1/0.057^2, 1/0.039^2, 1/0.053^2,$ $1/0.036^2,\ 1/0.019^2, 1/0.020^2)$.

Kun matriisin M arvot ovat oikeassa suuruusluokassa, tarkastellaan vielä kerran algoritmin etenemistä ja tehdään tarvittavat muutokset hienosäätöparametreihin ϵ ja L. Uusilla matriisin M arvoilla HMC:n hyväksymistodennäköisyys on 1. Algoritmi käy siis turhan tarkkaan läpi parametriavaruutta käyttäen siihen ylimääräistä laskenta-aikaa. Askeleen pituutta ϵ_0 voidaan siis pidentää. Toistamalla simulaatioita useaan kertaan pienillä ite-



Kuvio 8: Muuttujan α 500 ensimmäistä simuloitua arvoa HMC-algoritmilla, kun $M = \text{Diag}(1, ..., 1)$. Vasemmalla hienosäätöparametrit ovat $\epsilon_0 = 0.1$ ja $L_0 = 10$. Oikealla vastaavat arvot ovat 0.01 ja 100. Vasemmasta kuviosta huomataan, että simuloidut arvot pysyvät usein samassa kohtaa eli uuden ehdotuksen hyväksymistodennäköisyys on pieni. Oikealla hyväksymistodennäköisyys on suurempi.

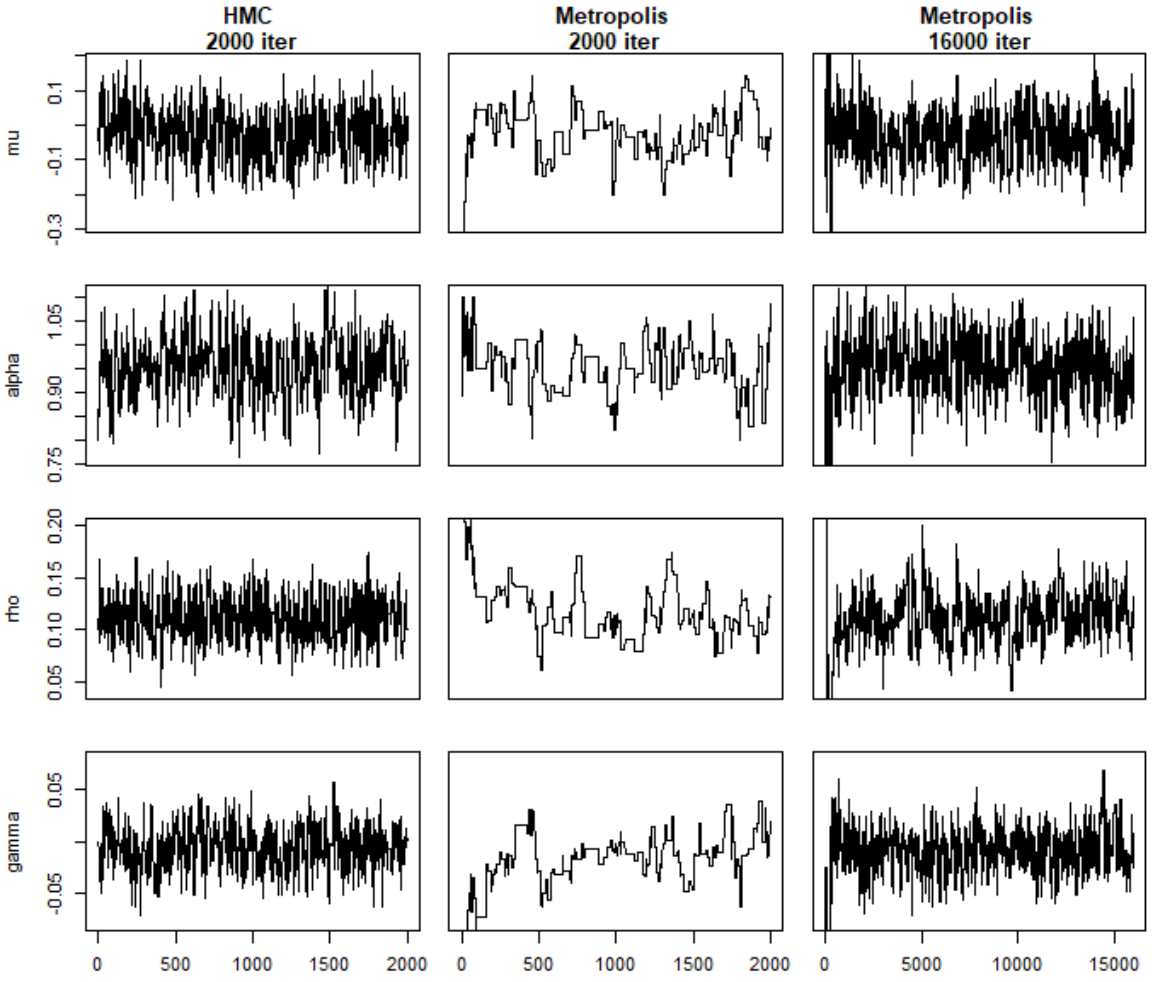
raatiomäärillä saadaan optimaaliseksi parametrin ϵ_0 arvoksi 0.025 ja asetetaan vastaavasti $L_0 = 40$.

Molemmilla algoritmeilla tehtiin kolme ketjua, joissa kaikissa oli 2000 iteraatiota. Ajallisesti HMC oli huomattavasti hitaampi kuin Metropolis-algoritmi: 2000 iteraatiossa HMC:n suoritusaika oli n. 900 sekuntia (n. 15 minuuttia), kun taas Metropolis-algoritmi suoritti saman 35 sekunnissa. Konvergoinnin tarkasteleminen kuitenkin osoittaa, että Metropolis-algoritmin kertoimet $\hat{R}$ eroavat selkeästi arvosta 1 (Taulukko 1). Sen sijaan HMC näyttää konvergoineen onnistuneesti (Taulukko 3). Esimerkiksi kuvion 9 otospoluista nähdään, että iteraatioiden lisääminen Metropolis-algoritmiin on tarpeen.

Kun Metropolis-algoritmissa kasvatetaan iteraatioiden määrää 16000 iteraatioon, konvergointikertoimien arvot $\hat{R}$ ovat samaa suuruusluokkaa kuin HMC:n vastaavat kertoimet ja otospolut näyttävät samankaltaisilta (kuvio 9). Yhden ketjun suoritusaika Metropolis-algoritmilla oli noin 170 sekuntia eli algoritmi oli noin viisi kertaa nopeampi kuin HMC.

Taulukoissa 2 ja 3 huomataan, että algoritmien tulosten välillä on eroja, kun tarkastellaan tehokkaita otoskokoja. Tasapuolisen vertailun vuoksi voidaan laskea tehokkaan otoskoon suhde simulointiin käytettyyn aikaan $n_{eff,T}$ kullekin muuttujalle. Keskimääräinen $n_{eff,T}$ HMC-algoritmille on noin 0.7, kun taas vastaava kerroin Metropolis-algoritmille on noin 4.8. Tämän perusteella Metropolis-algoritmi on noin 6.9 kertaa tehokkaampi kuin HMC.

Suoritusajan lisäksi algoritmeja voidaan vertailla autokorrelaation suhteen. Kuten voidaan olettaa, Metropolis-algoritmin parametrien autokorrelaatio on suurta: ketjun arvojen korrelaatio on selkeästi havaittavissa vielä 30 askeleen päästä (kuvio 10). HMC:ssä autokorrelaatio on pientä ja useimpien parametrien tapauksissa peräkkäisten arvojen korrelaatio on lähellä nollaa jo alle kymmenen askeleen päästä (kuvio 11).



Kuvio 9: Parametrien μ, α, ρ ja γ otospolut (malli A). Vasemmalla HMC-algoritmin muodostamat otospolut, keskellä Metropolis-algoritmin otospolut 2000 iteraatiolla ja oikealla Metropolis-algoritmin otospolut 16000 iteraatiolla.

Taulukko 1: Yhteenveto Metropolis-algoritmin simuloiduista arvoista (malli A, kolme ketjua, 2000 iteraatiota, $n_{burnin} = 1000$).

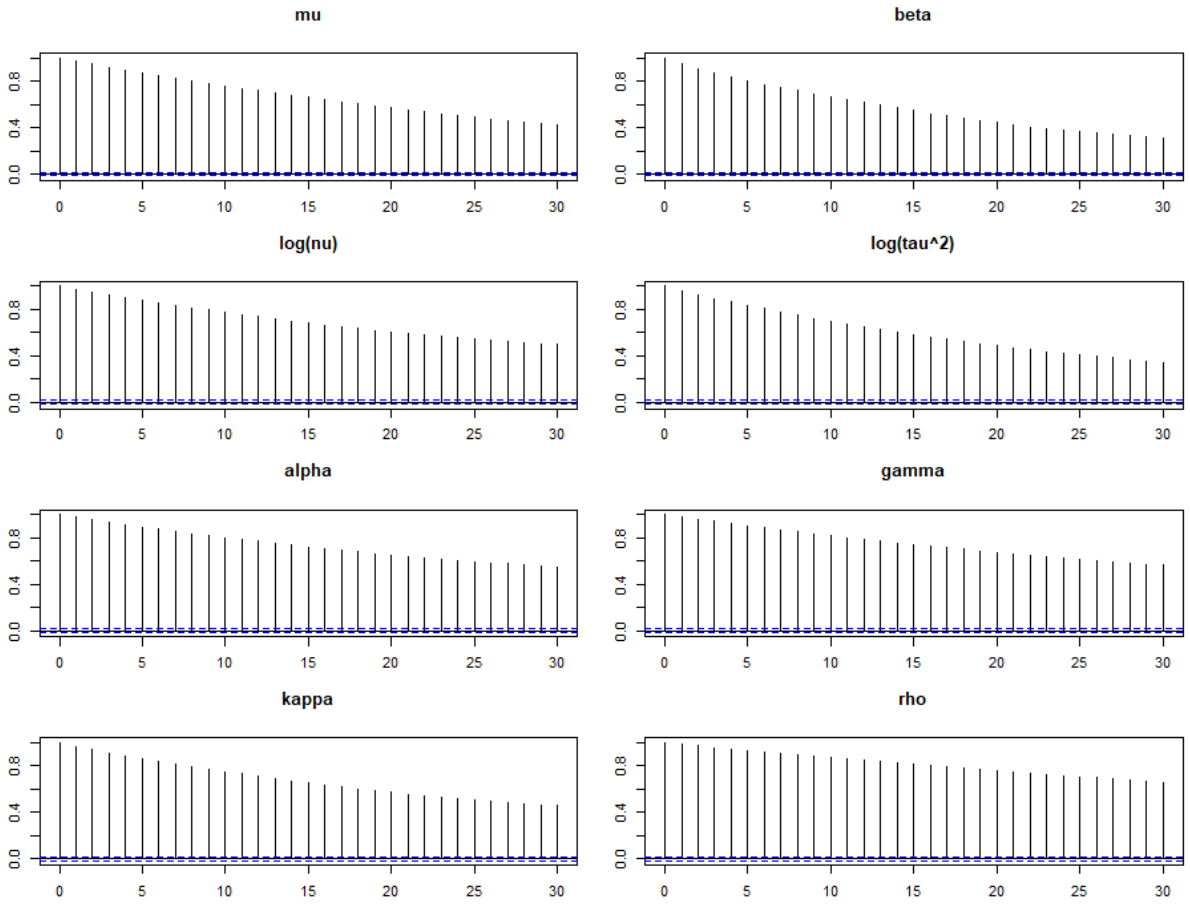
	keskiarvo	keskivirhe	keskihajonta	n_{eff}	$\hat{R}$
μ	-0.082	0.089	0.218	6	1.298
ν	3.964	0.093	0.506	30	1.102
α	0.914	0.041	0.131	10	1.272
κ	0.128	0.019	0.083	19	1.152
β	0.121	0.078	0.237	9	1.218
$\log(\tau^2)$	-2.449	0.0073	0.057	65	1.041
γ	-0.007	0.0021	0.026	125	1.046
ρ	0.078	0.049	0.110	5	1.643

Taulukko 2: Yhteenveto Metropolis-algoritmin simuloiduista arvoista (malli A, kolme ketjua, 16000 iteraatiota, $n_{burnin} = 8000$).

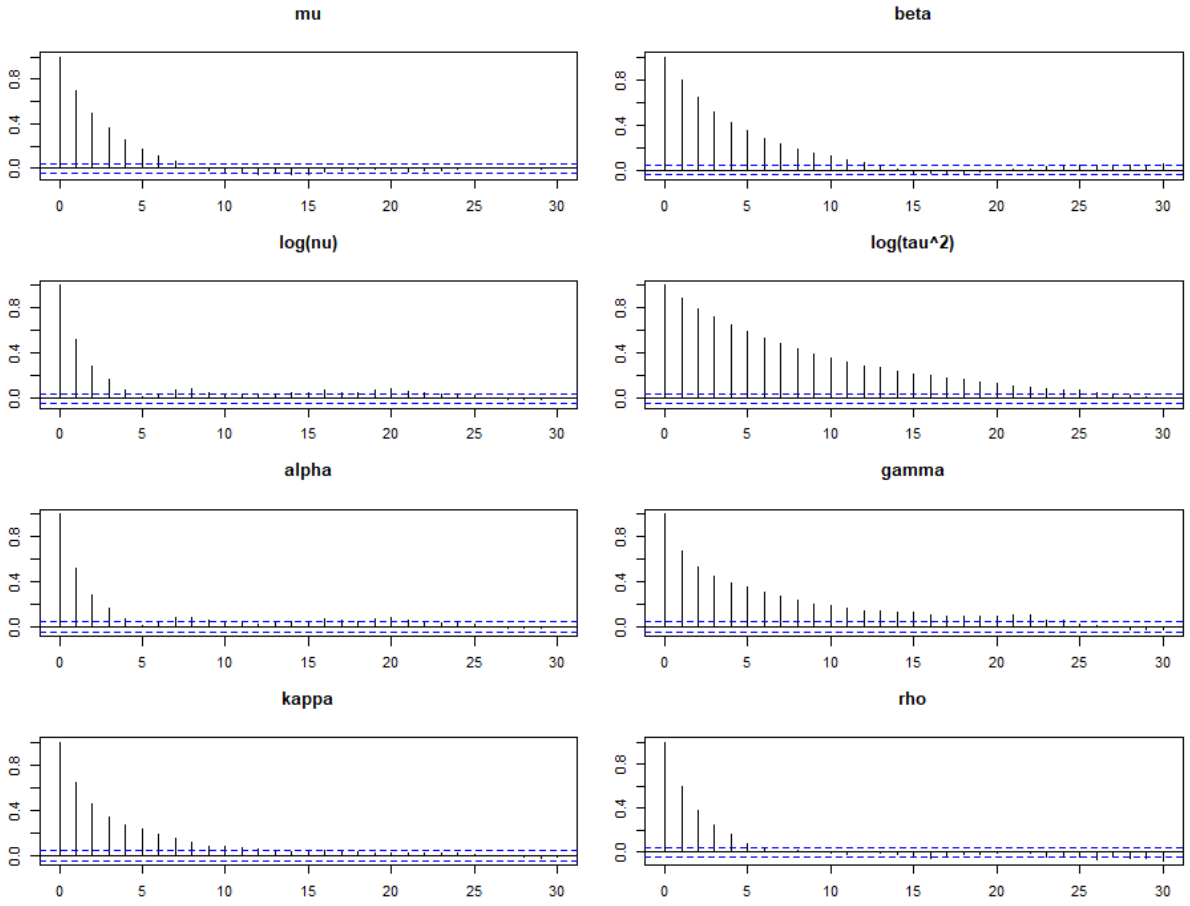
	keskiarvo	keskivirhe	keskihajonta	n_{eff}	$\hat{R}$
μ	-0.029	0.0026	0.065	609	1.004
$\log(\nu)$	4.077	0.012	0.333	807	1.006
α	0.953	0.0020	0.057	804	1.006
κ	0.110	0.0015	0.039	696	1.003
β	0.057	0.0012	0.053	1827	1.002
$\log(\tau^2)$	-2.449	0.0013	0.036	738	1.002
γ	-0.0074	0.00074	0.019	684	1.006
ρ	0.112	0.0010	0.020	414	1.004

Taulukko 3: Yhteenveto HMC-algoritmin simuloiduista arvoista (malli A, kolme ketjua, 2000 iteraatiota, $n_{burnin} = 200$).

	keskiarvo	keskivirhe	keskihajonta	n_{eff}	$\hat{R}$
μ	-0.027	0.0023	0.067	803	1.001
$\log(\nu)$	4.069	0.014	0.338	444	1.001
α	0.952	0.0027	0.058	443	1.001
κ	0.108	0.0016	0.042	549	1.006
β	0.058	0.0020	0.055	550	1.003
$\log(\tau^2)$	-2.449	0.0015	0.037	537	1.002
γ	-0.0074	0.00074	0.020	532	1.002
ρ	0.110	0.00089	0.020	852	1.007



Kuvio 10: Metropolis-algoritmin parametrien autokorrelaatiokuvaajat mallissa A.



Kuvio 11: HMC-algoritmin parametrien autokorrelaatiokuvaajat mallissa A.

4.3 Malli B

Mallissa B tavoitteena on jälleen saada ensin onnistunut ajo hienosäätöparametrien alkuarvoilla, minkä tuloksilla voidaan muuttaa matriisi M oikeaan suuruusluokkaan ja siten optimoida algoritmin nopeus. Hienosäätöparametrien alkuarvoilla algoritmi pysyy vain samassa aloituspisteessä eli hyväksymistodennäköisyys on nolla. Jälleen pienennetään yksittäisen pukkihyppyaskeleen pituutta siten, että asetetaan $\epsilon_0 = 0.001$ ja $L = 1000$. Näillä arvoilla saadaan onnistunut, vaikkakin hidas konvergoiminen. Näillä tuloksilla muokataan matriisi M oikeaan suuruusluokkaan.

Yrityksen ja erehdyksen kautta asetetaan $\epsilon_0 = 0.08$, jolloin hyväksymistodennäköisyydeksi saadaan noin 65%. Ehdon $\epsilon L = 1$ myötä asetetaan $L_0 = 12.5$, mutta tällöin silmämääräisesti tarkasteltuna ketjut eivät sekoitu tarpeeksi hyvin eli parametrin L_O tulisi olla suurempi. Tarkastelemalla otospolkuja parametrin L_0 eri arvoilla saadaan optimaaliseksi arvoksi $L_0 = 20$.

Lopullisessa vertailussa tarkastellaan ainoastaan aineiston y ensimmäistä tuhatta havaintoa algoritmien nopeuttamiseksi. HMC-algoritmin 2000 iteraation kierroksessa kesti noin 500 sekuntia, kun taas Metropolis-algoritmin vaatima aika 38000 iteraatiossa oli noin 170 sekuntia. Näillä iteraatiomäärillä algoritmien tulokset osoittautuivat samankaltaisiksi, kun tarkastellaan tehokkaita otoskokoja n_{eff} (taulukko 4 ja taulukko 5). Metropolis-algoritmi oli jälleen selvästi nopeampi kuin HMC. Tämä voi johtua osittain siitä, että sekä uskottavuusfunktio että prosessin h_t gradientti lasketaan rekursiivisesti, mikä kuluttaa paljon laskenta-aikaa HMC:ssa.

Tuloksien perusteella voidaan laskea kertoimet $n_{eff,T}$ kullekin parametrille. HMC-algoritmissa kyseisten kerrointen keskiarvo $\bar{n}_{eff,T} \approx 2.6$, kun taas vastaava keskiarvo Metropolis-algoritmille on noin 9.2. Metropolis-algoritmi on siis noin 3.5 kertaa tehokkaampi, kun tarkastellaan tehokkaita otoskokoja.

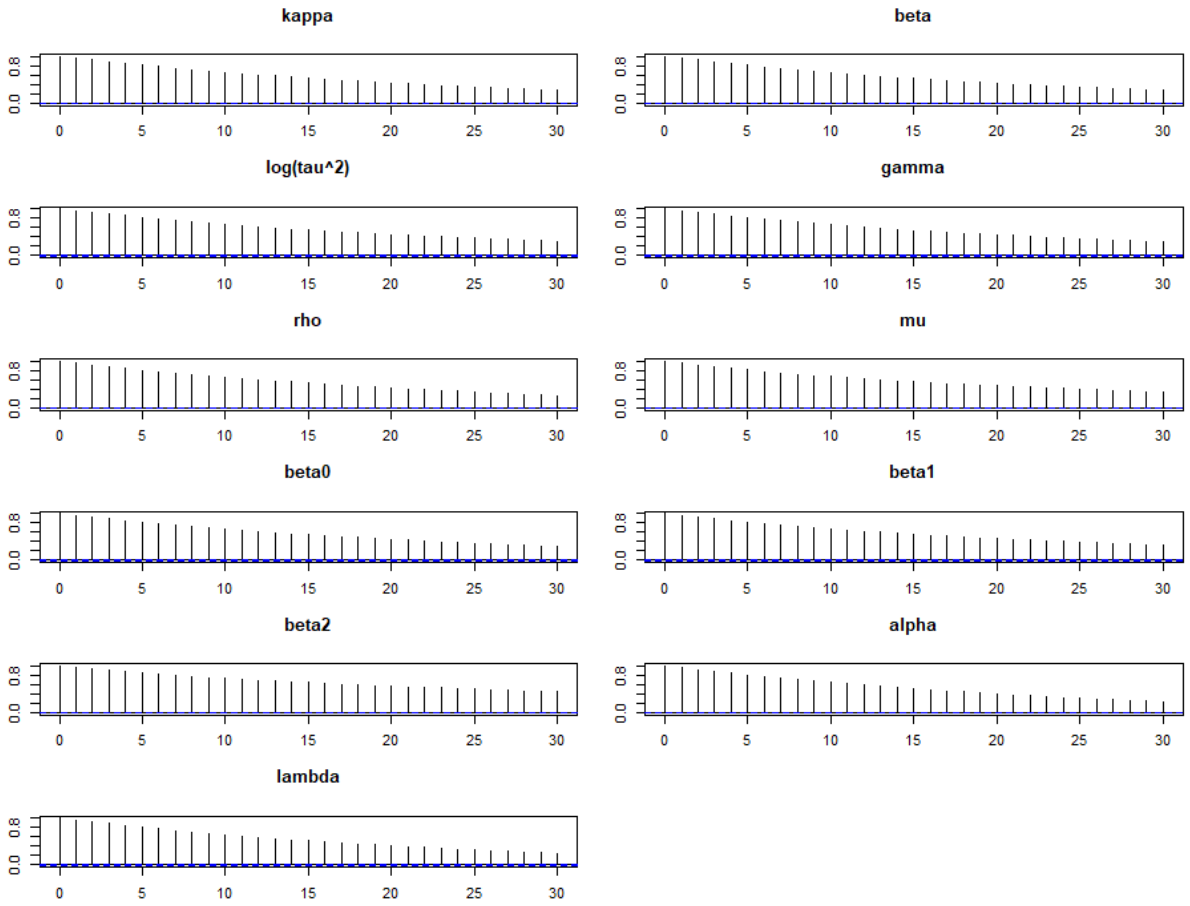
Metropolis-algoritmin ja HMC:n erot tulevat selvästi esille algoritmien autokorrelaatiokuvissa (kuvio 12 ja kuvio 13). Metropolis-algoritmissa korrelaatio on yhä havaittavissa, kun tarkastellaan 30 yksikön päässä olevia Markovin ketjun arvoja. HMC:ssä sen sijaan peräkkäisten havaintojen autokorrelaatio on lähellä nollaa jo 10. viiveellä useimpien parametrien tapauksessa. Sama ilmiö on havaittavissa molemmissa malleissa.

Taulukko 4: Yhteenveto Metropolis-algoritmin simuloiduista arvoista (malli B, kolme ketjua, 38000 iteraatiota, $n_{burnin} = 10000$).

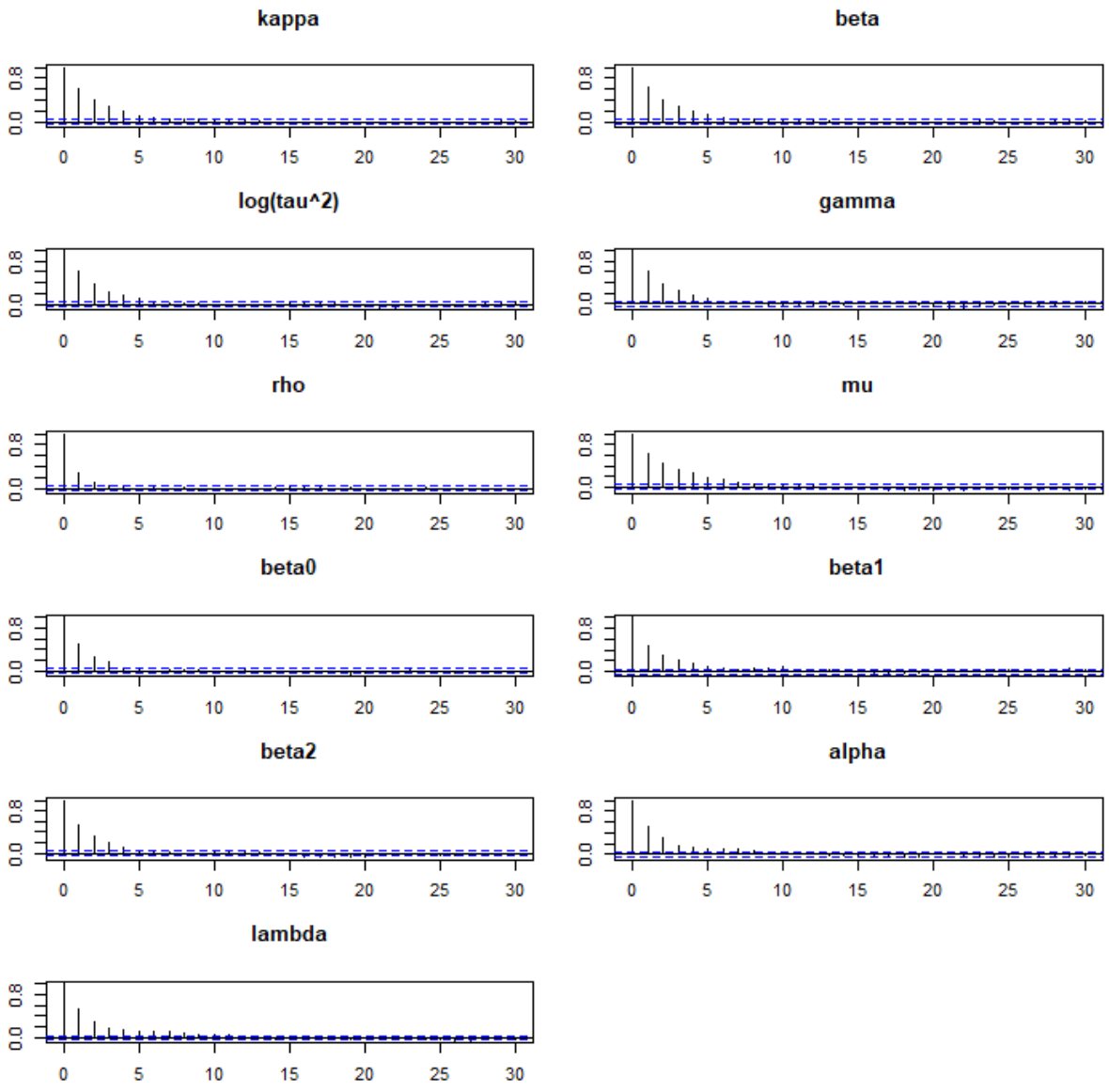
	keskiarvo	keskivirhe	keskihajonta	n_{eff}	$\hat{R}$
κ	0.756	0.0066	0.279	1766	1.001
β	1.572	0.015	0.612	1709	1.001
$\log(\tau^2)$	-6.409	0.0055	0.218	1582	1.001
γ	1.738	0.0031	0.125	1610	1.001
ρ	0.054	0.00053	0.023	1875	1.001
μ	1.607	0.0083	0.263	1000	1.002
β_0	$6.5 \cdot 10^{-7}$	$4.1 \cdot 10^{-9}$	$1.7 \cdot 10^{-7}$	1682	1.001
β_1	0.859	0.00045	0.017	1451	1.001
β_2	0.029	0.00017	0.0047	743	1.000
α	0.00069	0.000010	0.00045	1882	1.001
λ	-0.049	0.0018	0.080	1947	1.001

Taulukko 5: Yhteenveto HMC-algoritmin simuloiduista arvoista (malli B, kolme ketjua, 2000 iteraatiota, $n_{burnin} = 200$).

	keskiarvo	keskivirhe	keskihajonta	n_{eff}	$\hat{R}$
κ	0.759	0.0081	0.277	1183	1.001
β	1.581	0.018	0.607	1192	1.001
$\log(\tau^2)$	-6.419	0.0069	0.213	937	1.005
γ	1.744	0.0035	0.122	1198	1.004
ρ	0.053	0.00047	0.023	2334	1.000
μ	1.600	0.0087	0.250	829	1.001
β_0	$6.6 \cdot 10^{-7}$	$4.2 \cdot 10^{-9}$	$1.7 \cdot 10^{-7}$	1562	1.001
β_1	0.859	0.00048	0.017	1312	1.002
β_2	0.029	0.00014	0.0047	1205	1.001
α	0.00069	0.000012	0.00044	1378	1.002
λ	-0.049	0.0021	0.078	1351	1.003



Kuvio 12: Metropolis-algoritmin parametrien autokorrelaatiokuvaajat mallissa B.



Kuvio 13: HMC-algoritmin parametrien autokorrelaatiokuvaajat mallissa B.

5 Yhteenveto

MCMC-algoritmit ovat olleet tärkeä osa Bayes-tilastotiedettä jo vuosikymmeniä, mutta mallien monimutkaistuessa myös simulointialgoritmien tulee pysyä mukana kehityksessä. Vasta hiljattain Bayes-tilastotieteeseen implementoidun Hamiltonin Monte Carlo -algoritmin avulla pyritään ratkaisemaan suuridimensioisten mallien simulointiongelmat ja algoritmi onkin löytänyt tiensä tutkijoiden keskuuteen Stan-ohjelman kautta. Tästä huolimatta HMC:n idea on jäänyt monelle ammattilaisellekin tuntemattomaksi, mikä johtuu osittain algoritmin matemaattisesta monimutkaisuudesta. Tässä tutkimuksessa esiteltiin HMC-algoritmin rakenne ja syvennyttiin algoritmin hyviin ja huonoihin puoliin. Tämän lisäksi HMC:tä sovellettiin yleisiin finanssiaikasarjamalleihin. Vertailualgoritmina käytettiin Bayes-tilastotieteessä suosittua Metropolis-algoritmia.

HMC perustuu vahvasti Metropolis-algoritmiin, mutta muuttaa oleellisesti ehdotusjakaumaa: siinä missä Metropolis-algoritmin uusi ehdotus tulee satunnaisesti nykyisen pisteen ympäristöstä, HMC käyttää hyväkseen jakauman differentiaalirakennetta ja generoi uuden ehdotuksen alueelta, jossa tiheysfunktion arvot ovat suuria. HMC-algoritmi lainaa ideansa klassisesta mekaniikasta ja sen toimintaperiaatetta voidaankin kuvailla painovoimaesimerkin avulla, missä jakauman gradientti on kuin maapallon painovoimakenttä ja algoritmin eteneminen muistuttaa satelliitin liikerataa. HMC:n implementointi voi tuottaa vaikeuksia, sillä algoritmin hienosäätöparametrien valintaan ei ole mitään yleistä sääntöä ja usein nämä parametrit valitaankin "yrityksen ja erehdyksen kautta".

Tässä työssä tarkasteltaviksi malleiksi valittiin kaksi finanssimallia, joissa mallinnetaan osake- ja korkosijoitusten tuottoa. HMC-algoritmilla ja Metropolis-algoritmilla tuotettiin tarkkuudeltaan samankaltaiset tulokset molemmissa malleissa. Suuren hyväksymistodennäköisyyden takia HMC tarvitsi huomattavasti vähemmän iteraatioita kuin Metropolis-algoritmi. Tästä huolimatta Metropolis-algoritmi osoittautui selvästi nopeammaksi, kun tarkasteltiin laskentaan käytettyä aikaa. HMC-algoritmissa Markovin ketjun autokorrelaatio oli molemmissa malleissa selvästi pienempää kuin Metropolis-algoritmissa.

Tämän työn malleissa HMC osoittautui huomattavasti hitaammaksi kuin Metropolis-algoritmi. HMC:n on näytetty olevan nopeampi kuin Metropolis-algoritmi tilanteissa, joissa mallin parametrien välinen korrelaatio on suuri (Neal, 2011). Tällöin Metropolis-algoritmin tehokkuus kärsii pienen hyväksymistodennäköisyyden takia, kun taas HMC:n

toimintaan korrelaatiolla on vain pieni vaikutus. Myös kasvattamalla mallin parametrien määrää voidaan osoittaa HMC:n olevan tehokkaampi.

Kysyntä HMC:n kaltaisille algoritmeille kasvaa, kun käytettävät mallit monimutkaistuvat. Siksi onkin tärkeää tutkia HMC-algoritmin tehokkuutta erilaisilla malleilla ja moniulotteisissa tilanteissa. HMC:n erilaiset variaatiot, kuten NUTS ja RMHMC, ovat tärkeitä suunnannäyttäjiä tulevaisuuden algoritmeille ja onkin oleellista vertailla niitä muihin simulointialgoritmeihin niin tehokkuuden kuin käytännöllisyydenkin suhteen.

Lähteet

R. Abraham & J. E. Marsden. *Foundations of Mechanics.* Benjamin/Cummings Publishing Company, 1978.

C. Andrieu & J. Thoms. A Tutorial on Adaptive MCMC. *Statistics and Computing*, 18: 343–373, 2008.

I. Beichl & F. Sullivan. The Metropolis Algorithm. *Computing in Science and Engineering*, 2:65–69, 2000.

E. T. Bell. *Men of Mathematics.* WSOY, 1963.

M. Betancourt. A Conceptual Introduction to Hamiltonian Monte Carlo. *arXiv:1701.02434*, 2017.

G. E. P. Box & G. C. Tiao. *Bayesian inference in statistical analysis.* John Wiley and the sons, INC., 1992.

S. P. Brooks & A. Gelman. General Methods for Monitoring Convergence of Iterative Simulations. *Journal of Computational and Graphical Statistics*, 7:434–455, 1998.

P. Del Moral. Non Linear Filtering: Interacting Particle Solution. *Markov Processes and Related Fields*, 2:555–580, 1996.

P. Diaconis & D. Ylvisaker. Conjugate Priors for Exponential Families. *The Annals of Statistics*, 7:269–281, 1979.

S. Duane, A. D. Kennedy, B. J. Pendleton, & D. Rweth. Hybrid Monte Carlo. *Physics Letters B*, 195:216–222, 1987.

R. Engle & V. Ng. Measuring and testing the impact of news on volatility. *The Journal of Finance*, 48:1749–1778, 1993.

Finanssivalvonta. Henkivakuutukset, 10 2017. URL `http://www.finanssivalvonta.fi/fi/Finanssiasiakas/Tuotteita/Vakuutukset`.

N. Gatzert. Implicit options in life insurance: An overview. *Springler-Verlag*, 2009.

A. E. Gelfand & A. F. M. Smith. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, 85:398–409, 1990.

A. Gelman & D. B. Rubin. Inference from Iterative Simulation Using Multiple Sequences. *Statistical Science*, 7:457–511, 1992.

A. Gelman, J. Carlin, H. Stern, D. Dunson, A. Vehtari, & D. Rubin. *Bayesian Data Analysis*. Chapman and Hall, third edition, 2014.

P. Gilbert & R. Varadhan. *numDeriv: Accurate Numerical Derivatives*, 2016. URL `https://CRAN.R-project.org/package=numDeriv`. R package version 2016.8-1.

M. Girolami & B. Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society*, 73:123–214, 2011.

E. Hairer, C. Lubich, & G. Wanner. *Geometric Numerical Integration. Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, Berlin, 2006.

M. Hardy. *Investment Guarantees, Modeling and Risk Management for Equity-Linked Life Insurance*. John Wiley and Sons, Inc, 2003.

M. D. Hoffman & A. Gelman. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15:1593–1623, 2014.

S. Ip & J. Jewson. Hamiltonian Monte Carlo. Projekti, jossa vertaillaan HMC- ja Metropolis-algoritmia, Marraskuu 2010.

H. Ishwaran. Applivations of Hybrid Monte Carlo to Bayesian generalized linear models: Quasicomplete separation and neural networks. *Journal of Computational and Graphical Statistics*, 8:779–799, 1999.

B. Leimkuhler & S. Reich. *Simulating Hamiltonian Dynamics*. Cambridge University Press, Cambridge, 2004.

A. Luoma & A. Puustelli. Bayesian analysis of participating life insurance contracts with American-styleoptions. In *AFIR Colloquium*, München, 2009.

A. Luoma, A. Puustelli, & L. Koskinen. Bayesian analysis of equity-linked savings contracts with American-style options. *Quantitative Finance*, 2013.

D. J. C. MacKey. *Information Theory, Inference and Learning Algorithms.* Cambridge University Press, New York, 2003.

T. Marwala. *Condition Monitoring Using Computational Intelligence Methods.* Springer, 2012.

N. Metropolis & S. Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 44:335–341, 1949.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, & A. H. Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21, 1953.

R. M. Neal. *Bayesian Learning for Neural Networks.* PhD thesis, University of Toronto, 1995.

R. M. Neal. MCMC using Hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo*, pages 113–162. Chapman and Hall, 2010.

R. M. Neal. An Improved Acceptance Procedure for the Hybrid Monte Carlo Algorithm. *Journal of Computational Physics*, 111:194–203, 2011.

D. B. Nugraho & T. Morimoto. Estimation of Realized Stochastic Volatility Models using Hamiltonian Monte Carlo-Based Methods. *Computational Statistics*, 30:491–516, 2015.

C. O'Brien. The downfall of equitable life in the United Kingdom: The mismatch of strategy and risk management. *Risk Management and Insurance Review*, 2006.

R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2013. URL `https://www.R-project.org`.

C. Robert & G. Casella. A Short History of Markov Chain Monte Carlo: Subjective Recollections from Incomplete Data. *Statistical Science*, 26:102–115, 2011.

Stan development team. Rstan: the r interface to stan, 2017. URL `http://mc-stan.org`. R package version 2.16.2.