TEKNILLINEN KORKEAKOULU
Tietotekniikan osasto
Informaatiotekniikka

Tomas Ukkonen

Hajautettu Monte Carlo simulaatio paikkatietoanalyysissä

Diplomi-insinöörin tutkintoa varten tarkastettavaksi jätetty diplomityö

Masala, 8. maaliskuuta 2006

Työn valvoja	Professori Samuel Kaski
Työn ohjaaja	Professori Tapani Sarjakoski

TEKNILLINEN KORKEAKOULU Tietotekniikan osasto	DIPLOMITYÖN TIIVISTELMÄ

Tekijä	Päiväys
Tomas Ukkonen	6.3.2006
	Sivumäärä 74

Työn nimi
Hajautettu Monte Carlo simulaatio paikkatietoanalyysissä

Professuuri	Koodi
Informaatiotekniikka	T-61

Työn valvoja
Professori Samuel Kaski

Työn ohjaaja
Professori Tapani Sarjakoski

Diplomityö käsittelee epävarmuutta sisältävien korkeusmallien valuma-alueiden laskentamenetelmiä. Työssä perehdyttiin hilamuotoisten korkeusmallien valuma-alue-analyysiin, tilastollisiin korkeusmallin epävarmuuden mallintamistapoihin ja Monte Carlo simulaatioon.

Olemassa olevien menetelmien pohjalta kehitettiin hajautettuun laskentaan sopivat laskenta-algoritmit, joiden toimintaa ja skaalautuvuutta testattiin Tieteellisen laskennan CSC:n klusterilaskentaympäristössä. Simulaation tulosten tarkkuuden arvioimiseksi johdettiin korkeaulotteiselle aineistolle ja simulaatiomenetelmässä lasketuille valuma-aluekartoille sopiva konvergenssin arviointimenetelmä.

Diplomityö koostuu seitsemästä luvusta. Johdannossa esitellään työn taustaa ja käsitteitä, sekä kuvataan työssä ratkaistut ongelmat. Työn pohjana oleva aiempi tutkimus ja sen tulokset esitellään luvussa kaksi. Työn toteutusosassa johdetaan aluksi menetelmä Monte Carlo simulaation tarkkuuden arvioimiselle simulaatiokierrosten määrän kasvaessa. Luvussa neljä esitetään ratkaisut valuma-aluelaskennan hajauttamiseksi ja arvioidaan algoritmien tehokkuutta. Luvussa viisi käydään tarkemmin läpi työssä tehdyn laskentaohjelman teknisempiä yksityiskohtia. Ohjelman toimintaa ja skaalautuvuutta testattiin oikealla korkeusmalliaineistolla. Luvussa kuusi käsitellään nämä laskentaohjelman ja menetelmien toimintaan liittyvät mittaustulokset ja vertailut. Lopuksi yhteenveto-osassa käydään läpi työn keskeiset tulokset ja jatkokehitysmahdollisuudet.

Avainsanat

Valuma-alue analyysi, hajautettu laskenta, Monte Carlo simulaatio, prosessikonvoluutio

HELSINKI UNIVERSITY OF TECHNOLOGY Department of Computer Science and Engineering	ABSTRACT OF MASTER'S THESIS

Author	Date
Tomas Ukkonen	March 6, 2006
	Pages 74

Title of thesis	
Parallel Monte Carlo Simulation in Spatial Analysis	
Professorship Computer and Information Science	Professorship Code T-61
Supervisor Professor Samuel Kaski	
Instructor Professor Tapani Sarjakoski	

The focus of this master's thesis is on computational methods for calculating drainage areas from digital elevation models (DEM) including uncertainty. The work concentrates on drainage area analysis, statistical methods for modelling uncertainness in DEMs, Monte Carlo simulation and on parallel computing methods.

Based on earlier work in DEM-based catchment delineation methods, drainage area algorithms for distributed computing environments were developed and tested on the computer cluster of Scientific Computing CSC. To ascertain accuracy and validity of simulation results, a new method suited for high dimensional data and drainage area calculation was developed to estimate convergence of calculated drainage area maps.

The thesis consists of seven chapters. Chapter 1 describes the background of this work, important concepts and solved problems. Earlier findings upon which this thesis is based are described in greater detail in Chapter 2. The convergence estimation method for the simulation described in Chapter 3 is the first independent finding of this study. Developed parallel algorithms for drainage basin analysis are then described and discussed in Chapter 4, which is the main theoretical part of the thesis. Chapter 5 describes further technical details as well as the design of a developed computer program. The scalability and correctness of the developed computer program were tested with real world data. Chapter 6 describes and discusses these tests and measurements. The final chapter summarises the results of the thesis and puts forward proposals for future work.

Keywords

Drainage area analysis, parallel computing, Monte Carlo simulation, process convolution

Esipuhe

Tämä työ on tehty Geodeettisella laitoksella geoinformatiikan ja kartografian osastolla. Työssä on sovellettu osaston korkeusmallitutkimusten tuloksia. Hajautettu Monte Carlo simulaatio paikkatietoanalyysissä diplomityön on ohjannut professori, osastonjohtaja Tapani Sarjakoski. Kiitän häntä mielenkiintoisesta aiheesta, työstä, sekä ohjauksesta. Diplomityötä on valvonut professori Samuel Kaski Teknillisestä korkeakoulusta. Häntä kiitän ohjauksesta, työn välivaiheiden palautteesta, sekä muista neuvoista. Geodeettisen laitoksen vanhempaa tutkijaa Juha Oksasta kiitän geostatistiikkaan ja valuma-alue laskentaan liittyvistä neuvoista ja opastuksesta. Lisäksi kiitän osaston muuta väkeä hyvästä yhteistyöstä ja mukavasta työilmapiiristä.

Masalassa 8. maaliskuuta 2006

Tomas Ukkonen

Sisältö

Kuvat

Taulukot

Merkintöjä

μ_x, $\hat{\mu_x}$	satunnaismuuttuja x:n jakauman keskiarvo ja keskiarvon estimaatti
σ_x^2, $\hat{\sigma}_x^2$	satunnaismuuttuja x:n jakauman varianssi ja varianssin estimaatti
$\gamma(\mathbf{h})$	satunnaismuuttuja semivarianssi $\frac{1}{2}Var[\mathbf{z}(\mathbf{x}) - \mathbf{z}(\mathbf{x}+\mathbf{h})]$
$\mathcal{N}(\mu_\mathbf{x}, \mathbf{\Sigma_x})$	vektoriarvoinen normaalijakauma keskiarvolla $\mu_\mathbf{x}$ ja kovariassinmatriisilla $\mathbf{\Sigma_x}$
$\mathcal{N}(\mathbf{\Pi_X}, \mathbf{\Sigma_X}, \mathbf{\Psi_X})$	matriisinormaalijakauma keskiarvolla $\mathbf{\Pi_X}$ ja kovarianssimatriiseilla $\mathbf{\Sigma_X}$ ja $\mathbf{\Psi_X}$
$W_\nu(\mathbf{S})$	Wishart matriisijakauma parametrillä $\mathbf{S}$ ja vapausasteella ν
χ_ν^2	Ksiin neliön jakauma vapausasteella ν
$F_{n,\nu}$	Fisherin F-jakauma vapausasteilla n ja ν
$mF_{n,\nu}$	jakauma, jossa satunnaismuuttuja X/m on $F_{n,\nu}$-jakaantunut
$tr(\mathbf{A})$	matriisin jälki, $tr(\mathbf{A}) = \sum_i \mathbf{A}(i,i)$
$vec(\mathbf{A})$	matriisin vektorointi sarakkeittan $\left[\begin{array}{cccccc} a_{11} & .. & a_{N1} & a_{12} & .. & a_{NN} \end{array} \right]$
$\mathbf{A} \otimes \mathbf{B}$	Kroneckerin tulo matriisien $\mathbf{A}$ ja $\mathbf{B}$ välillä
$\delta(\mathbf{h})$	Diracin delta funktio, $\delta(\mathbf{0}) = 1$ ja 0 muulloin
$\mathcal{F}(\cdot)$	diskreetti Fourier-muunnos

Luku 1

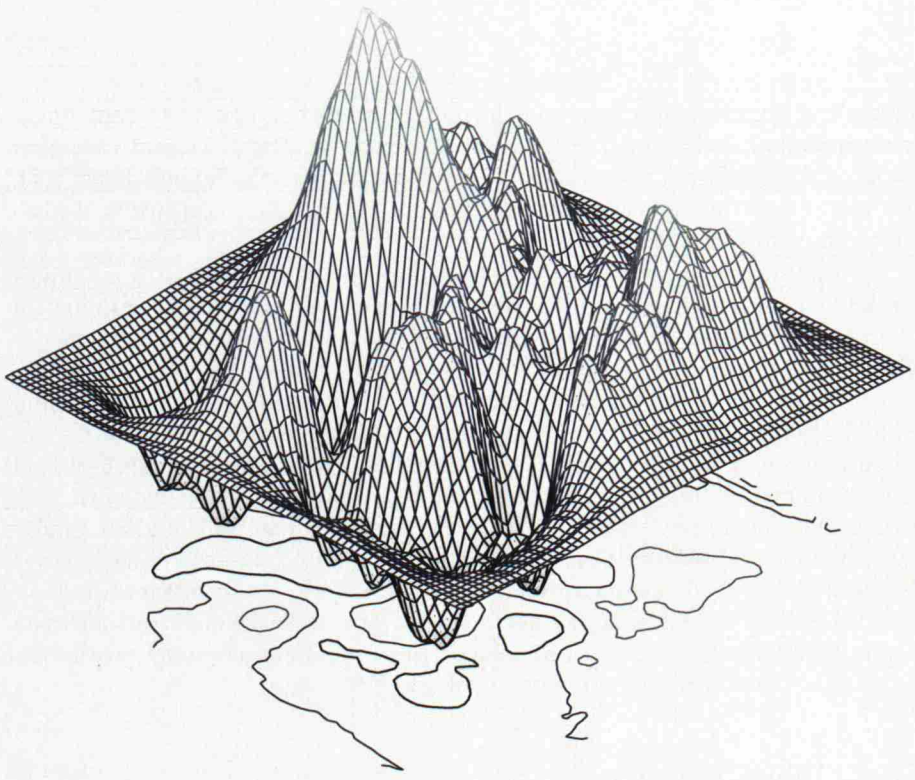
Johdanto

Digitaaliset korkeusmallit ovat geoinformatiikassa yleisesti käytetty tapa kuvata maanpinnan korkeutta. Digitaalinen korkeusmalli (DEM, Digital Elevation Model) voidaan esittää esimerkiksi suorakaiteen muotoisena hilana (Kuva 1.1), jota käsitellään tietokoneella kaksiulotteisena, maanpinnan korkeuksia sisältävänä taulukkona (korkeusmatriisi).

Digitaaliset korkeusmallit ovat yksinkertaisuutensa takia suosittu maanpinnan korkeuden esitystapa, kun korkeustietoa halutaan analysoida, korjailla tai muuten jälkikäsitellä. Tyypillisiä korkeusmallien käsittely ja analysointimenetelmiä ovat kvalitatiivisten mittausvirheiden ja vääristymien poisto kuten kuoppien täyttö, pisteen valuma-alueen laskeminen korkeusmallista , korkeusmallin kolmiulotteinen visualisointi ja maastosta mitattujen arvojen inter- tai ekstrapolointi muihin mallin pisteisiin. Perusmenetelmien lisäksi on edistyksellisempiä tilastollisia mallintamis- ja analysointimenetelmiä käytetty spatiaalisten eli etäisyyden mukaan riippuvien mittauksien, esimerkiksi korkeusmittauksien, analysoimisessa ja tutkimisessa [Swa 99, Ker 00]. Tilastolliset mallit ja niillä laskevat menetelmät kuten myöhemmin esiteltävä Monte Carlo (MC) integrointi eli simulointi ja satunnaismuuttujien näytteistys ovat sekä laskennallisesti aiempaa huomattavasti raskaampia että matemaattisesti monimutkaisempia vaatien aiempaa tehokkaampien laskentamenetelmien kehittämistä.

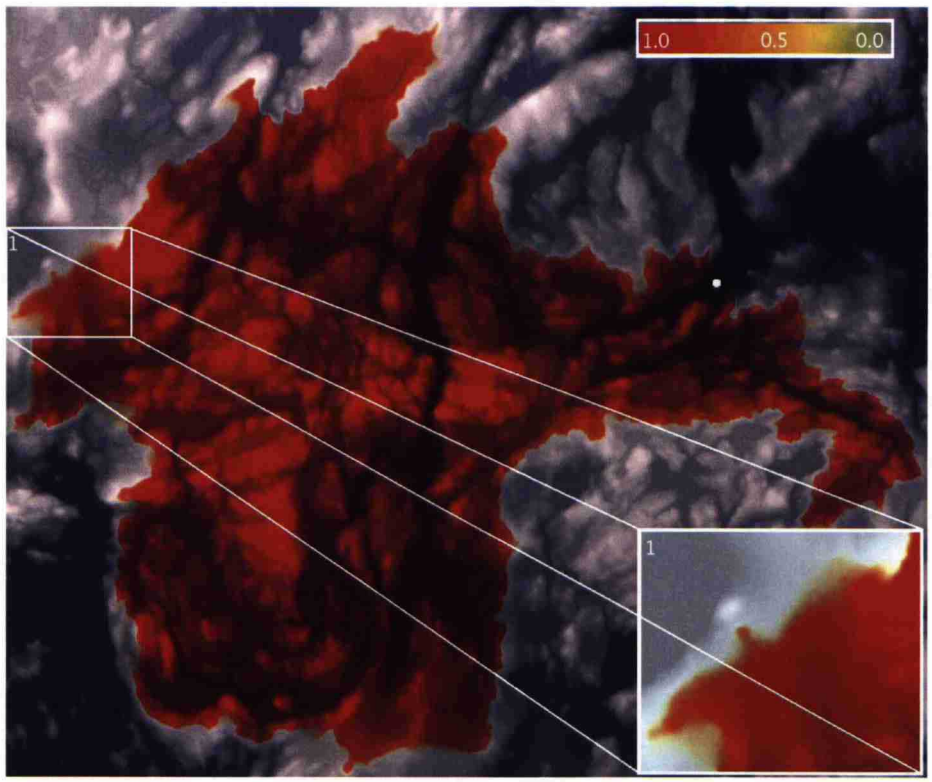
1.1 Työn tavoite

Työn tavoitteena on tutustua korkeusmallien, jotka tässä työssä ovat aina suorakaiteen muotoisia korkeusmatriiseja $\mathbf{X}$, tilastollisiin mallintamistapoihin ja valuma-alueiden laskentamenetelmiin, sekä kehittää ja toteuttaa hajautettuja valuma-alueiden laskentamenetelmiä. Työn lopputuloksena on korkeushilan tilastollista jakaumamallinnusta, Monte Carlo integrointia ja hajautettua laskentaa hyödyntävä valuma-alueiden laskentaohjelma, jonka toimintaa ja skaalautuvuutta testataan Tieteellisen laskennan CSC:n klusterilaskentaympäristössä. Ohjelman käyttäjä voi määritellä korkeusmallille normaalijakaumaan perustuvan virhejakauman ja laskea valuma-alueet merkitsemillensä valumapisteille. Ohjelma toimii iteratiivisesti parantaen ratkaisun tarkkuutta joka iteraatiolla ja haluttaessa ohjelma pystyy iteraatioiden välissä laskemaan arvioita tuloksessa vielä jäljellä olevasta virheestä. Työssä käytettävät laskentamenetelmät pe-



Kuva 1.1: Diskretoitu korkeusmalli

rustuvat pääasiassa Geodeettisella laitoksen korkeusmallien tilastollisia virhemalleja ja valuma-alueiden laskentaa käsitteleviin tutkimuksiin[Oks 05, Oks 6].



Kuva 1.2: Todennäköisyys kuulua pisteen valuma-alueeseen ja suurennos alueen 1 reunalta

1.1.1 Laskentamenetelmä

Toteutettava ohjelma laskee annetun korkeusmallin korkeusmatriisin $\mathbf{X}$ jokaiseen pisteeseen $\mathbf{x}$ todennäköisyyden p, jolla piste kuuluu annetun pisteen $\mathbf{y}$ valuma-alueeseen. Pisteen $\mathbf{y}$ valuma-alue korkeusmatriisissa $\mathbf{X}$ on veden virtausalue, joka virratessaan ulos matriisin $\mathbf{X}$ alueelta kulkee pisteen $\mathbf{y}$ kautta. Se lasketaan yleensä hyvin suoraviivaisesti simuloimatta veden todellisia virtauksia. Yksinkertaisimillaan pisteen $\mathbf{y}$ valuma-alue voidaan määritellä niiksi korkeusmallin pisteiksi, joista löytyy laskeva reitti pisteeseen $\mathbf{y}$. Käytännössä asia ei kuitenkaan ole näin yksinkertainen. Valuma-aluelaskennassa täytetään myös korkeusmatriisin $\mathbf{X}$ kuopat. Tämän seurauksena veden virtaukset eivät jää jumiin kartan keskiosiin, mikä on havaittu huomattavasti parantavan laskettujen valuma-alueiden oikeellisuutta.

Mikäli korkeusmatriisi $\mathbf{X}$ ei sisällä epävarmuutta, kuuluu kukin korkeusmallin piste yksikäsitteisesti pisteen $\mathbf{y}$ valuma-alueeseen eli $p = 0 \vee p = 1$. Korkeusmatriisia $\mathbf{X}$ ei kuitenkaan tiedetä tarkasti. Siinä on satunnaisvirhettä $\mathbf{E}$, jonka

vaikutuksia valuma-alueiden rajauksiin halutaan tutkia.

Merkitään funktiolla $\mathbf{f_y}(\mathbf{X}) \rightarrow \mathbf{D}$ laskentaa, jossa korkeusmatriisista $\mathbf{X}$ lasketaan pisteen $\mathbf{y}$ valuma-alue matriisiin $\mathbf{D}$, jossa $\mathbf{D}(\mathbf{x}) = 1$, kun piste $\mathbf{x}$ kuuluu pisteen $\mathbf{y}$ valuma-alueeseen ja $\mathbf{D}(\mathbf{x}) = 0$, kun se ei kuulu siihen. Kun virhetermi $\mathbf{E}$ mallinnetaan tilastollisesti satunnaismuuttujana voidaan kehitettävän ohjelman laskennan lopputulos, eli todennäköisyysmatriisi $\mathbf{P}$ pisteiden kuulumisesta pisteen $\mathbf{y}$ valuma-alueeseen kirjoittaa odotusarvona

$$\mathbf{P} = E[\mathbf{f}(\mathbf{X} + \mathbf{E})] = \int \mathbf{f_y}(\mathbf{X} + \mathbf{E}) p(\mathbf{E} \mid \alpha) d\mathbf{E}.$$

Kaavassa $p(\mathbf{E} \mid \alpha)$ on satunnaisvirheen jakauma, jonka parametrit α ohjelman käyttäjä voi asettaa korkeusmallissa olevan virheen mukaisiksi tai kokeilla erillaisten virhemallien vaikutuksia laskettavaan valuma-alueeseen. Parametreillä, jotka ovat kappaleessa kaksi esiteltävien geostatistiikan käsitteiden ja menetelmien mukaisia, voidaan vaihdella virhepinnan $\mathbf{E}$ pisteiden välistä korrelaatiota $\mathbf{\Sigma_E}(\alpha)$. Myöhemmin esiteltävään, aiempaan tutkimukseen pohjautuen sarakkeittain vektoroidun virhepinnan $vec(\mathbf{E})$ jakaumana käytetään normaalijakaumaa ja se on muotoa $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma_X}(\alpha))$. Riippumattomia, eri tavoin korreloituneita virhepintoja voidaan myös haluttaessa lisätä karttaan useita, jolloin $\mathbf{E} = \sum \mathbf{E}_i$.

Odotusarvon laskemiseksi täytyy ohjelman laskea odotusarvo $E[\mathbf{f}(\mathbf{X} + \mathbf{E})]$ eli integraali

$$\int \mathbf{f_y}(\mathbf{X} + \mathbf{E}) p(\mathbf{E} \mid \alpha) d\mathbf{E}$$

riittävän tarkasti. Tähän käytetään suoraa normaalijakautuneiden muuttujien generointimenetelmää. Näennäisestä helppoudestaan huolimatta ongelma on laskennallisesti hankala ja normaalijakautuneet muuttujat generoidaan luvussa kaksi esiteltävän prosessikonvoluution ja diskreetin Fourier muunnoksen avulla. Parametrisoinnin mukaisien satunnaismuuttujien $\mathbf{E}$ realisaatioita käyttämällä voidaan odotusarvon integraalia arvioida keskiarvolla

$$\mathbf{P}_K = \frac{1}{K} \sum \mathbf{f_y}(\mathbf{X} + \mathbf{E}_i).$$

Tällöin on tulosten luotettavuuden kannalta tärkeää pyrkiä arvioimaan lasketun keskiarvon virhettä, jotta tulosten luotettavuudesta voitaisiin olla varmoja. Tätä varten kuvataan kappaleessa kaksi perusmenetelmänä virheen monitorointimenetelmä, johon perustuen kehitetään yllä kuvatulle laskennalle ja suurille valuma-aluekartoille soveltuva virheen arviointimenetelmä.

Visualisointi laskennan tuloksesta $\mathbf{P}_K$ on nähtävissä kuvassa 1.2. Valuma-alue on laskettu valkoiselle pisteelle $\mathbf{y}$ (suurennettu visualisointi syistä) ja väriarvot punaisesta keltaiseen kuvaavat todennäköisyyksiä (1..0), minkä mukaan pisteet kuuluvat valuma-alueeseen. Oikeassa alanurkassa on suurennos valuma-alueen reunasta.

1.2 Sisältö

Luvussa kaksi kuvataan olemassa olevat menetelmät, algoritmit ja tulokset valuma-alueiden analysoimisesta, korkeusmallin virheen tilastollisesta mallintamisesta, satunnaismuuttujien generointimenetelmistä, Monte Carlo integroinnista ja sen tarkkuuden arvioinnista eli konvergenssista sekä hajautetusta laskennasta.

Luvussa kolme luvun kaksi konvergenssinmonitorointimenetelmä yleistetään toimimaan useassa ulottuvuudessa ja laskentaa muutetaan skaalautumaan paremmin korkeaulotteisille valuma-aluekartoille. Tämän jälkeen luvussa neljä kehitetään hajautetun laskennan algoritmit valuma-aluelaskentaa varten. Kehitetyt menetelmät pohjautuvat luvun kaksi algoritmeihin ja ideoihin.

Esiteltyjen ratkaisujen pohjalta kehitetyn laskentaohjelman toimintaa ja teknisiä ratkaisuja käsitellään luvussa viisi. Työssä ohjelmasta kehitettiin kaksi eri versiota. Hajauttamattomia, kappaleessa kaksi kuvattuja algoritmejä käyttävä ohjelma ja hajautettuja algoritmejä käyttävä ohjelma. Luvussa kuusi esitellään tulokset, jotka vertailevat eri algoritmien toimintaa keskenään, konvergenssinarviointimenetelmän tarkkuutta ja hajautuksen skaalautuvuutta suurille prosessorimäärille. Lisäksi esitellään kuvia laskennan tuloksista. Tehtyjen mittaustulosten perusteella tehdään johtopäätökset laskentamenetelmien tehokkuudesta ja toiminnasta.

Viimeisessä lopetuskappaleessa kerrataan työn keskeiset tulokset ja havainnot, sekä esitellään jatkokehitysideoita menetelmien parantamiseksi.

Luku 2

Taustaa

Valuma-alueiden laskenta suurista korkeusmalleista on raskas laskennallinen operaatio, koska järkevien tuloksien saamiseksi korkeusmallien kvalitatiivisia virheitä tulee laskennan aikana korjailla. Valuma-alue laskennassa tälläisiä laadullisia virheitä ovat korkeushiloissa ilmenevät kuopat, jotka estävät vettä valumasta alueen reinoilla ja joiden täyttäminen parantaa laskettujen valuma-alueiden oikeellisuutta. Työssä tutustuttiin kahteen kuoppientäyttömenetelmään. Näistä vanhempi ja hitaampi on Jensonin/Dominguen [Jen 88] menetelmä ja uudempi Liun/Wangin [Wan 05] laskentamenetelmä.

Korkeusmalleissa olevien virheiden tilastollisen mallinnuksen ja parametrisoinnin osalta olennainen käsite on spatiaalisen korrelaation esittäminen semivariogrammilla [Ole 99]. Kun semivariogrammi on saatu määriteltyä, voidaan mm. Geodeettisella laitoksella tutkitut [Oks 05] korkeushilojen tilastolliset mallinnustavat esittää yksityiskohtaisesti.

Laskennan toteutuksen kannalta on olennaista tehdä Monte Carlo integroinnissa [Rob 04] tarvittavat normaalijakaantuneiden virhepintojen generointi nopeasti. Tämä voidaan tehdä digitaalisen signaalinkäsittelyn prosessikonvoluutio ja Fourier-muunnos menetelmillä [Mit 98, Swa 99].

Tulosten tarkkuuden arvioimiseksi on tarvetta pystyä arvioimaan Monte Carlo integroinnin tulosten tarkkuutta ns. monitorointimenetelmällä, josta aluksi esitellään yksiulotteisille muuttujille soveltuva menetelmä [Rob 04]. Esiteltyä tulosta laajennetaan myöhemmin työssä toimimaan useammilla muuttujilla - esimerkiksi lasketuilla valuma-alueilla. Esiteltyjen algoritmien hajauttamiseksi on lisäksi tarpeen esitellä hajautetun ja rinnakkaislaskennan käsitteitä [And 99].

2.1 Valuma-alueiden laskentamenetelmiä

Fysikaalisesti oikea ratkaisu veden virtauksien laskemiseksi korkeusmallista olisi ratkaista tai simuloida veden virtausta nesteitä kuvaavien osittaisdifferentiaaliyhtälöiden mukaan, missä reunaehtoina olisivat mallin korkeusarvot. Käyttötarkoituksen kannalta riittävän hyviä ratkaisuja voitaisiin todennäköisesti saada simuloimalla asiaan liittyviä osittaisdifferentiaaliyhtälöitä elementtimenetelmillä. Hyviä lähtökohtia tälläisille approksimaatioille voisivat tarjota hydrologian ja veden virtauksien osittaisdifferentiaaliyhtälöt (esim. Navier-Stokes).

Valuma-alueiden laskentamenetelmät ratkaisevatkin ongelman samalla ta-

voin kuin elementtimenetelmät käyttämällä jatkuvan korkeusmallin sijasta tasavälisesti diskretoitua korkeusmallia Muita yhtäläisyyksiä varsinaisten elementtimenetelmien kanssa ei oikeastaan ole. Varsinaisen simuloinnin sijaan veden virtaamia maanpinnalla arvioidaan heuristisesti pinnan gradientin avulla, mikä on katsottu antavan valuma-alueiden laskennassa riittävän realistisia tuloksia. Lisäksi näin laskenta on nopeampaa kuin veden virtauksia tarkemmin simuloitaessa. Mikäli menetelmiä kuitenkin haluttaisiin tulevaisuudessa parantaa realistisempaan suuntaan ottamalla huomioon esimerkiksi maanpinnan sisäiset tai alaiset virtaukset, voisivat elementtimenetelmät tarjota hyvän tavan virtaamien tarkempaan laskemiseen. Tämä vaatisi kuitenkin parempien laskentamenetelmien ja suuremman laskentakapasiteetin lisäksi myös tarkempia, kolmiulotteisia mittauksia maanpinnasta ja maan koostumuksesta.

2.1.1 Jensonin ja Dominguen algoritmit

Korkeusmallipohjaisten valuma-alueiden laskennan perusmenetelminä ovat yhä käytössä olevat, pääasiassa S. K. Jensonin ja J. O. Dominguen 1980-luvulla [Jen 88] kehitettämät laskentamenetelmät. Jensonin ja Dominguen kehittämiä menetelmiä ovat mm.

- virtaussuuntien laskeminen,
- kuoppien täyttö,
- valumien kertymien laskenta ja
- valuma-alueiden laskenta.

Virtaussuuntien laskenta

Korkeusmallien virtaussuuntien laskennassa etsitään jokaiselle alueen pisteelle hyvä ja jatkoanalyysin kannalta käyttökelpoinen nesteen, esimerkiksi veden, virtauksen suunta. Laskennassa maanpinnan oletetaan yksinkertaisuussyistä olevan jotain kovaa, nestettä läpäisemätöntä ainetta, esimerkiksi kiveä.

Virtaussuunnat voidaan laskea suoraviivaisesti normaaleissa tapauksissa, joissa pisteen ympäristöstä löytyy yksikäsitteinen, kaikkia muita naapuruston pisteitä matalampi piste. Tällöin laskentapisteellä voidaan määritellä olevan alaspäin suuntautunut, yksikäsitteinen diskreetti "gradientti". Olkoon korkeusmallin korkeusmatriisi $\mathbf{F}$, tällöin virtausgradientin suuruus $(x_0, y_0) \longrightarrow (x_1, y_1)$ määritellään seuraavasti:

$$g = \frac{\mathbf{F}(x_1,y_1)-\mathbf{F}(x_0,y_0)}{\|(x_1-x_0)^2+(y_1-y_0)^2\|}.$$

Rasteroiduissa korkeusmatriiseissa on kuitenkin pisteitä, joille tälläistä pistettä ei löydy. Tällöin virtaussuuntaa ei voida määrittää tai se ei ole yksikäsitteinen. Piste voi esimerkiksi olla koko naapurustoa matalampi kuoppa, se voi sijaita tasaisella alueella, tai naapuruston pisteistä monet ovat saman korkuisia, jolloin hyviä virtaussuuntia on useita.

Koska ongelmallisiin laskentapisteisiin ei ole muodostunut esimerkiksi järveä, tulee veden kuitenkin virrata jonnekin ja virtaussuunta halutaan määrittää kvalitatiivisesti järkevästi korkeusmallissa olevista kvantitatiivisista virheistä huolimatta. Virtaussuunnat tulisi määrittää lisäksi niin, että virtaamat eivät muodosta silmukoita. Näiden ongelmien takia virtaussuuntien laskenta on varsinkin isoille korkeusmalleille raskas laskennallinen ongelma.

Dominguen kehittämässä virtaussuuntien laskenta-algoritmissä lähdetään siitä, että korkeusmalli on kuopaton. Tällöin voidaan löytää kaikille pisteille silmukaton valumapolku alueen reunalle. Näin ongelmaksi jää vain käsitellä tapaukset, joissa paras gradientti on nolla tai yhtäsuuria pienimpiä gradientteja on useita.

Tasaisten alueiden nollagradientti kohdat voidaan ratkaista etsimällä naapurustosta nollagradienttipisteitä, joille virtaussuunta on saatu jo laskettua. Asettamalla virtaussuunta vain pisteisiin, joiden virtaussuunta on tiedossa, saadaan lopulta käsiteltyä kaikki nolla gradienttipisteet.

Jos pienimpiä gradientteja on useita, asetetaan virtaussuunta heuristisesti joko kohti keskiarvon mukaista suuntaa tai satunnaisesti johonkin mahdollisista virtaussuunnista. Koska Domingue on julkaisussaan epätäsmällinen selittäessään heuristisen menetelmänsä toimintaa, työssä toteutetussa algoritmissa virtaussuunnan valinta toteutettiin niin, että algoritmin toiminta on sama kaikissa Dominguen paperissaan [Jen 88] antamissa esimerkeissä. Työn toteutuksessa virtaussuunta valittiin satunnaisesti mikäli mahdollisista virtaussuunnista laskettujen valumien varianssivektorin pituus ($\sqrt{\sigma_x^2 + \sigma_y^2}$) kasvoi heuristista kynnysarvoa suuremmaksi.

Taulukko 2.1: Virtaussuuntien laskenta-algoritmi

```
Funktio Valumat = LaskeValumat(Korkeusmalli)
  Joukko Määrittelemättömät
  Jokaiselle pisteelle b Korkeusmallin reunalla
     Valumat[b] = UlospäinSuunta(b)
  LopetaJokaiselle

  Jokaiselle Korkeusmallin sisäpisteelle p
    Joukko Suunnat, Pisteet
    (Suunnat, Pisteet) = Etsi virtaussuunnat p:n ympäristöstä

    Jos |Suunnat| == 0 ∨ Tiputus(Suunnat) < 0.0
      Valumat[p] = 0 ;; suuntaa ei voida määrittää
    Muutoin Jos |Suunnat| == 1
      Valumat[p] = Suunnat[1]
    Muutoin Jos Tiputus(Suunnat) > 0.0 ;; monta virtaussuuntaa
      Vektori ms = E[Suunnat]
      Vektori vs[1..N] = ∛tr(Σ_Suunnat)

      Jos |vs| < 1.0 ∧ |ms| > 0.8
        Valumat[p] = LaskeSuunta(ms) ;; keskiarvon mukaisesti

      ;; ei luoda virtauksia ylämäkeen
```

```
        Jos Korkeusmalli[p] < Korkeusmalli[p + Valuma[p]]
          Valumat[p] = ValitseSatunnaisesti(Suunnat)
        LopetaJos
      Muutoin
        Valumat[p] = ValitseSatunnaisesti(Suunnat)
      LopetaJos
    Muutoin
      Määrittelemättömät.lisää(p, Suunnat)
    LopetaJos
  LopetaJokaiselle

  ;; käsitellään tasaisella alueella olevat pisteet
  Joukko Uudet
  Toista Jos |Määrittelemättömät| > 0
    Joukko Suunnat, Piste p

    Jos |Uudet| > 0
      (p,Suunnat) = Uudet.poista()
    Muutoin
      (p,Suunnat) = Määrittelemättömät.poistaSatunnaisesti()
    LopetaJos

    Jokaiselle v Suunnat joukossa
      n = p + v

      Jos Korkeusmalli[p] >= Korkeusmalli[n]
        Jos n ∉ Määrittelemättömät
          Valumat[p] = v
          Uudet.lisää(Naapurit(p), EtsiSuunnat(Naapurit(p)))
          Määrittelemättömät.poista(Naapurit(p))
          Suunnat = ∅ ;; lopetetaan laskentapisteen osalta
        LopetaJos
      LopetaJos
    LopetaJokaiselle

  Jos Suunnat ≠ ∅
    ;; suuntaa ei saatu ratkaistuksi
    Määrittelemättömät.lisää(p, Suunnat)
  LopetaJos

  LopetaToista
LopetaFunktio
```

Valumien kertymien laskenta

Kun virtaussuunnat on laskettu, voidaan valumien kertymät laskea yksinkertaisesti. Valumien kertymät lasketaan lähtemällä liikkeelle jokaisesta alueen pisteestä ja seuraamalla laskettua virtaussuuntaa mikäli sellainen on pisteelle olemassa. Algoritmi laskee kertymät erilliseen taulukkoon, joka on aluksi asetettu

nolliksi ja virtaussuuntia seuratessaan se kasvattaa jokaisen vieraillun pisteen arvoa yhdellä. Erityisesti on huomattava, että Jensonin/Dominguen määrittelemä valumien kertymiä laskeva algoritmi ei kasvata aloituspisteen arvoa lainkaan. Tällöin pisteet, joiden läpi ei kulje lainkaan virtausta, saavat arvon nolla.

Taulukko 2.2: Valumien kertymien laskenta

```
Funktio Kertymät = LaskeKertymät(Valumat)
  Jokaiselle korkeusmallin pisteelle p
    Piste n = p + Valumat[p]

    Toista jos n on korkeusmallin alueella
      Kertymät[n] = Kertymät[n] + 1
      n = n + Valumat[n]
    LopetaToista
  LopetaJokaiselle
LopetaFunktio
```

Jensonin ja Dominguen kuoppientäyttöalgoritmi

Korkeusmallin kuoppien täyttäminen tarpeellinen operaatio virtaussuuntien laskemiseksi. Ideaalisesta korkeusmallista pitäisi jokaiselle pisteelle löytyä ei-nouseva polku alueen reunoille. Tämän saavuttamiseksi kuopat tulisi täyttää kuopan matalimman reunapisteen tasolle l. niin, että kuopasta voi olla virtausta ulospäin ja jatkaa kunnes kuoppia ei enää ole jäljellä. Näin vesi voi valua ulos alueelta jäämättä korkeusmallissa oleviin kuoppiin. Alkuperäisen korkeusmatriisin $\mathbf{Z}$ kuopat on täytetty oikein, kun kuopaton korkeusmatriisi $\mathbf{W}_f$ täyttää seuraavat ehdot [Pla 01]:

1. $b_1(\mathbf{W}_f) : \forall \mathbf{p} \mathbf{W}_f(\mathbf{p}) \geq \mathbf{Z}(\mathbf{p})$
2. $b_2(\mathbf{W}_f)$: $\forall \mathbf{p} \exists \{\mathbf{x}_i\}$: $\mathbf{x}_0 = \mathbf{p}$, $\mathbf{x}_N \in \partial \mathbf{W}_f$, $\|\mathbf{x}_i - \mathbf{x}_{i-1}\|^2 < 2$, $\mathbf{W}_f(\mathbf{x}_i) \geq \mathbf{W}_f(\mathbf{x}_{i-1})$
3. $b_3(\mathbf{W}_f) : \nexists \mathbf{V} \neq \mathbf{W}_f$ s.e. $[b_1(\mathbf{V}) \wedge b_2(\mathbf{V}) \wedge (\forall \mathbf{p} : \mathbf{V}(\mathbf{p}) \leq \mathbf{W}_f(\mathbf{p}))]$,

missä $\partial \mathbf{W}_f$ on korkeusmallin reunapistejoukko.

Jensonin/Dominguen kuoppientäyttöalgoritmissä täytetään aluksi aina yksittäiset, yhden solun kokoiset kuopat, joiden tunnistaminen ja täyttäminen on suoraviivaista. Tämän jälkeen korkeusmallista etsitään yhtenäiset valuma-alueet, jotka virtaavat yhteiseen alueen pisteeseen, eli ovat sisällä samassa kuopassa. Myös yhteiseen alueen reunapisteeseen tai ulos virtaavat alueet yhdistetään laskennan ajaksi yhtenäiseksi valuma-alueeksi. Kun erilliset valuma-alueet on saatu laskettua, etsitään kullekin valuma-alueelle matalin purkupiste jokaiseen naapurivaluma-alueiseen.

Kuoppiin liittyvien tietorakenteiden luonnin jälkeen algoritmi täyttää valuma-alueita satunnaisessa järjestyksessä. Algoritmi valitsee valuma-alueesta matalimman kynnyskorkeuden purkupisteen ja etenee sen mukaiseen naapurivaluma-alueeseen. Tällä tavoin edetään valuma-alueiden matalimpia purkupisteitä pit-

kin eteenpäin kunnes saavutaan alueen reunalle tai kunnes päädytään jo aiemmin vierailtuun valuma-alueeseen eli löydetään silmukka. Algoritmi käsittelee silmukat yhdistämällä silmukassa olevat valuma-alueet yhdeksi ja nostamalla silmukan pisteet silmukan valumareitin korkeimman kynnyskorkeuden tasolle. Silmukan yhdistämisen jälkeen laskentaa jatketaan normaalisti. Alueen reunalle päädyttäessä nostetaan aloitus valuma-alueen pisteet lasketun valumapistepolun korkeimman kynnyskorkeuden tasolle.

Kuoppientäyttöalgoritmissa tulee aluksi laskea kuopat eli valuma-alueet ja niiden reunat. Kuoppien reunat voidaan määrittää laskemalla virtaussuunnat kaikille mahdollisille pisteille aiemmin esitetyn virtaussuuntien laskenta-algoritmin mukaisesti. Koska nyt kuitenkin korkeusmallissa on kuoppia, ei kaikille korkeusmallin pisteille löydy ulosvirtaussuuntaa. Nämä pisteet ovat kuoppien valuma-alueiden kertymäpisteitä ja niitä voidaan käyttää aloituspisteinä valuma-alueiden laskennassa. Myös valumienkertymä algoritmiä voidaan käyttää korkeusmalliin, jossa kaikille pisteille ei ole määritelty ulosvirtaussuuntaa. Laskemmalla valumien kertymät korkeusmatriisissa voidaan tunnistaa kuoppien reunat, joiden läpi ei kulje virtausta. Yhtenäiset valuma-alueet löydetään lisäämällä kuoppien aloituspisteeseen kaikki korkeusmallin pisteet, joista löytyy ei-nolla kertymiä sisältävä polku aloituspisteeseen.

Jensonin/Dominguen algoritmi on tietorakenteittensa ja toimintansa puolesta ratkaistavaan, melko yksinkertaiseen ongelmaan nähden monimutkainen. Se lähtee ratkaisemaan ongelmaa ihmismäisesti tunnistamalla aluksi kuopat ja täyttämällä niitä. Tämän seurauksena valuma-alue tietorakennetta ja sen kuoppia joudutaan käymään läpi ja päivittämään useita kertoja. Jensonin/Dominguen kuoppientäyttöalgoritmin suora toteutus on hidas korkeusmalleilla, joissa on paljon pieniä kuoppia. Jos esimerkiksi kaikki kuopat ovat sisäkkäisiä ja kuoppia lähdetään täyttämään aina sisimmästä kuopasta ulospäin, on läpikäytyjen valuma-aluepolkujen yhteispituus $O(D^2)$, kun alueella on kuoppia D kappaletta:

$$\sum_{i=1}^{D-1}(D-i) = O(D^2).$$

Vastaava aikakompleksisuus liittyy myös paikallisten valuma-alueiden purkupisteiden laskentaan, mikäli naapurivaluma-alueiden määrä kasvaa valuma-alueiden kokonaismäärän suhteen likimain lineaarisesti $O(D)$, joudutaan valuma-alueiden purkupisteitä laskemaan $O(D^2)$ kappaletta.

Jensonin/Dominguen esittelemä alkuperäinen algoritmi on melko primitiivinen ja sitä ei ole optimoitu erityisen hyvin. Algoritmi voisi täyttää kaikki seurattavalla polulla olevat valuma-alueet kerralla, jolloin menetelmä nopeutuisi huomattavasti. Tätäkin parempi tapa olisi lähteä täyttämään kuoppia aina reunoilta olevista valuma-alueista lähtien, joiden matalin valumapiste on korkeusmallin reunalla, ja edetä yksi valuma-alue kerrallaan mallin sisäosiin. Näin varsinaisen kuoppientäyttöosan laskennallinen kompleksisuus on $O(D)$, mikä voi silti olla hyvinkin suuri, mikäli kuoppien määrä on verrannollinen korkeusmallin kokoon NM, eli

$$O(D) \sim O(NM).$$

Tämä ei kuitenkaan ratkaise ongelmaa purkupisteiden laskennan ja käsittelyn osalta, jonka aikakompleksuus on luokkaa

$$O(D^2) \sim O(N^2 M^2).$$

Tätäkin ongelmaa voidaan kiertää etsimällä valuma-alueista vain matalin purkupiste ja yhdistämällä matalimman purkupisteen valuma-alueet kerran, jonka jälkeen sekä kuopat että purkupisteet täytyy laskea uudelleen, mutta kuoppien määrä on saatu puolitettua. Jensonin/Dominguen algoritmin parantelu ei kuitenkaan ole erityisen mielekästä, koska kuoppientäytölle on olemassa nopeampia ja elegantimpia menetelmiä. Dominguen algoritmi on monivaiheisuudessaan monimutkainen ja sen hajauttaminen voi olla hankalaa.

Taulukko 2.3: Jensonin ja Dominguen kuoppientäyttöalgoritmi

```
Funktio Korkeusmatriisi = TäytäKuopat(Korkeusmatriisi)
  ;; täytetään yksittäiset kuopat
  Jokaiselle korkeusmallin pisteelle p
    N(p) = { x | ||x - p|| < 2 }
    Jos ∀x ∈ N(p) : Korkeusmatriisi(p) < Korkeusmatriisi(x)
      Korkeusmatriisi[p] = min(Korkeusmatriisi[N])
    LopetaJos
  LopetaJokaiselle

  Valumat = LaskeValumat(Korkeusmatriisi)
  Kertymät = LaskeKertymät(Valumat)

  ;; lasketaan kuoppaisen korkeusmallin valuma-alueet
  ;; Sinks == kuoppien kertymäpisteet
  Sinks = {x | Valumat[x] == 0 }

  Jokaiselle s ∈ Sinks
    Alue[s] = LaskeValumaAlue(Valumat, s)
    Reunat[s] = { p ∈ Alue[s] | N(p) ∩ Alue[s] ≠ N(p) }
  LopetaJokaiselle

  Jokaiselle s ∈ Sinks
    Jokaiselle t ∈ Sinks, s ≠ t
      Jokaiselle sp ∈ Reunat[s]
        RR = N(sp) ∩ N(Reunat[t])
        Jokaiselle x ∈ RR
          korkeus = max(Korkeusmalli[sp], Korkeusmalli[x])
          Jos Vuotokohta_s[t].korkeus > korkeus
            Vuotokohta_s[t].korkeus = korkeus
            Vuotokohta_s[t].pisteet = { sp, x }
          LopetaJos
        LopetaJokaiselle
      LopetaJokaiselle
    LopetaJokaiselle
  LopetaJokaiselle

  ;; nostetaan ja yhdistetään valuma-alueita kunnes
```

```
;; kaikilla valuma-alueilla on vuotokohta alueen reunan yli
Joukko NonSinks

Toista jos |Sinks| > 0
  Valitse s ∈ Sinks

  Jos min(Alueenreunat ∩ Alue[s]) ≤ min(Vuotokohta_s[·].korkeus)
    Sinks.poista(s)
    NonSinks.lisää(s)
  Muutoin ;; yhdistä toisen joukon kanssa
    out = min_index(Vuotokohta_s[·].korkeus)
    Korota(s, Vuotokohta_s[out].korkeus)

    ;; yhdistetään valuma-alue out alueeseen ja
    ;; päivitetään samalla Vuotokohta taulukko
    Yhdistä(s, out, Vuotokohta)
    Sinks.poista(s)
  LopetaJos
LopetaToista

LopetaFunktio
```

Valuma-alueiden laskenta

Annetun pisteen tai pistejoukon valuma-alueet voidaan laskea seuraavan algoritmin mukaisesti. Toinen tapa valuma-alueiden laskemiseksi olisi laskea kartan virtaussuunnat ja laajentaa valuma-aluetta korkeusarvojen sijaan virtaussuuntien avulla.

Taulukko 2.4: Valuma-alueidet laskenta

```
Funktio ValumaAlue = LaskeValumaAlue(Korkeusmatriisi, Pisteet)
  Joukko B = Pisteet
  ValumaAlue = Pisteet

  Toista jos | B | > 0
    Piste p = B.poistaPiste()

    Jokaiselle naapuripisteelle n
      Jos Korkeusmatriisi[p] ≤ Korkeusmatriisi[n] ja
          n ∉ ValumaAlue
        B.lisääPiste(n)
        ValumaAlue.lisääPiste(n)
      LopetaJos
    LopetaJokaiselle
  LopetaToista

LopetaFunktio
```

Menetelmässä valuma-alueen pistejoukkoa kasvatetaan jo lasketun joukon reunoilta kunnes valuma-alueeseen ei lisätä enää uusia pisteitä. Algoritmin hajauttaminen ei vaikuta erityisen hankalalta, vaikkakin tällöin joukot A ja $valumaAlue$ täytyy hajauttaa eri laskentasolmujen kesken.

2.1.2 Wangin ja Liun algoritmi

Esitellyistä algoritmeistä kuoppientäyttövaihe on kaikkien hitain. Esiteltyä Jensonin/Dominiguen laskentamenetelmää ei parannettu 1980-luvulla tai 90-luvulla. Planchon ja Darboux [Pla 01] saivat 2001 aikaan olennaisen parannuksen kuoppien täyttöön, minkä jälkeen laskentaa saatiin nopeutettua vielä lisää. Liu esitti 2005 väitöskirjassaan [Wan 05] uuden, yksinkertaisen ja samalla tehokkaan algoritmin. Algoritmi on helppo ymmärtää ja toteuttaa ja sen toiminta perustuu tehokkaaseen prioriteettijono tietorakenteeseen [Knu 98, Man 05].

Wangin/Liun kuoppientäyttöalgoritmi ratkaisee kuoppientäyttöongelman yksinkertaisella tavalla pysyen kuitenkin laskennallisesti tehokkaana. Menetelmän teoreettinen aikavaativuus $N \times M$ kokoisella korkeusmallille on luokkaa

$$O(NMlog(NM)).$$

Algoritmissä vieraillaan jokaisessa korkeusmallin $N \times M$ pisteessä kerran ja tehdään prioriteettijonoon jokaisen korkeusmallin pisteen osalta yksi lisäys- ja poisto-operaatio ($O(log(P))$. Jos oletetaan piirin olevan vielä käsittelemättä olevan alueen neliöjuuren kokoinen $P \approx \sqrt{A}$, saadaan laskenta-ajaksi arvio

$$2\sum_{k=0}^{A-1} log\sqrt{A-k} = log\left[\prod_{k=0}^{A-1}(A-k)\right] = log(A!).$$

Käyttämällä nyt tähän Stirlingin approksimaatiota $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ tulee laskenta-ajaksi

$$log(A!) \approx \tfrac{1}{2}log(A) + A(log(A) - 1) = O(Alog(A)).$$

Menetelmän ideana on lähteä etenemään korkeusmallin reunoilta sisäänpäin. Aluksi kaikki alueen reunapisteet lisätään pistejoukkoon, jonka jälkeen joukosta valitaan piste p, jonka laajentaminen johonkin, ei vielä joukossa olevaan naapuripisteeseen n, aiheuttaa mahdollisimman pienen korkeuden noston pisteessä n. Laajentamisoperaatiossa pisteen n uudeksi korkeudeksi asetetaan

$$max(korkeus(p), korkeus(n))$$

ja piste n lisätään pistejoukkoon. Varsinaisessa algoritmissä pisteiden korotus tehdään ja lasketaan lisättäessä pisteitä reunajoukkoon. Tämä ei kuitenkaan muuta algoritmin toiminnan oikeellisuutta. Pisteen p korkeus $korkeus(p)$ on laskettu tässä vaiheessa oikein, joten oikea pisteen n korotus voidaan laskea jo tässä vaiheessa.

Taulukko 2.5: Wangin kuoppientäyttöalgoritmi

```
Funktio Korkeusmatriisi = TäytäKuopat(Korkeusmatriisi)
  Joukko KäsitellytPisteet
  PrioriteettiJono UudetPisteet   ;; reunan ulkopisteet,
                                  ;; joihin voidaan edetä
  Jokaiselle korkeusmatriisin reunapisteelle b
    Korotus[b] = Korkeus[b]
    UudetPisteet.lisää(Korotus[b], b)
  LopetaJokaiselle

  Toista jos |UudetPisteet| > 0
    p = UudetPisteet.poistaMatalin()
    KäsitellytPisteet.lisää(p)

    Jokaiselle p:n naapuripisteelle n
      Jos n ∉ UudetPisteet ja n ∉ KäsitellytPisteet
        Korotus[n] = Maksimi(Korkeus[n], Korotus[p])
        UudetPisteet.lisää(Korotus[n], n)
      LopetaJos
    LopetaJokaiselle
  LopetaToista

LopetaFunktio
```

Menetelmä voidaan näyttää toimivan oikein, eli täyttävän aiemmin esitetyt kouppientäyttöalgoritmin lopetusehdot. Todistus menetelmän oikeellisuudesta annetaan myöhemmin algoritmin hajautusta käsittelevässä luvussa. Menetelmän teoreettinen aikavaativuus on Jensonin/Dominguen algoritmia parempi, mikä on nähtävissä myös mittaustuloksista (Taulukko 2.6).

Taulukko 2.6: Kuoppientäyttöalgoritmien nopeudet [Wan 05]

Korkeus	Leveys	Domingue	Planch	Wang
500	500	30 s	1 s	1 s
2000	4000	1563 s	94 s	48 s
4000	4000	12971 s	651 s	210 s
8000	8000	59878 s	1516 s	456 s

Yksinkertaisen toteutuksensa takia myös algoritmin hajauttaminen vaikuttaisi olevan Jensonin/Dominguen algoritmia helpompaa.

2.2 Tilastollisia menetelmiä

2.2.1 Geostatistiikkaa

Geostatistiikka [Ole 99] on varsinaisesta tilastotieteestä eriytynyt geologisia mittauksia käsittelevä tieteenhaara. Se keskittyy analysoimaan mittauksia lähinnä

kaksiulotteisessa tasossa. Aikasarja-analyysin ja tilastollisen signaalinkäsittelyn [Hay 96] tavoin alan perusmenetelmät perustuvat toisen asteen statistiikkoihin l. korrelaatioihin, normaalijakaumaoletuksiin ja neliöllisen virheen minimointiin. Ala on kehitettynyt varsinaisesta tilastotieteestä erillisenä alana, minkä takia se käyttää paikoin normaalista tilastomatematiikasta poikkeavia käsitteitä ja merkintöjä. Menetelmien tyypillisinä tavoitteina on tilastollinen inter- tai ekstrapolaatio mittausalueen mittauspaikkojen välillä.

Näitä tilastollisia neliöllisen virheen minimoivia interpolaatiomenetelmiä kutsutaan geostatistiikassa *kriging*-menetelmiksi, joista tunnetuimmat ovat yksinkertainen, tavallinen ja universaali kriging (simple kriging, ordinary kriging, universal kriging). Menetelmät etsivät todennäköisimmän arvon uudelle alueen pisteelle, kun tiedetään alueen pisteiden välinen, etäisyyden funktiona muuttuva kovarianssi

$$Cov(\mathbf{h}) = Cov[\mathbf{z}(\mathbf{x}), \mathbf{z}(\mathbf{x}+\mathbf{h})].$$

Tällöin mittauksille $\mathbf{z}_2$ ja tuntemattomille pisteille $\mathbf{z}_1$ voidaan määritellä yhteinen normaalijaukama

$$\begin{bmatrix} \mathbf{z}_1 & \mathbf{z}_2 \end{bmatrix}^T \sim \mathcal{N}(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \mathbf{\Sigma_{11}}(\mathbf{h}) & \mathbf{\Sigma_{12}}(\mathbf{h}) \\ \mathbf{\Sigma_{21}}(\mathbf{h}) & \mathbf{\Sigma_{22}}(\mathbf{h}) \end{bmatrix})$$

Mallin perusteella voidaan laskea analyyttisesti ehdollinen jakauma

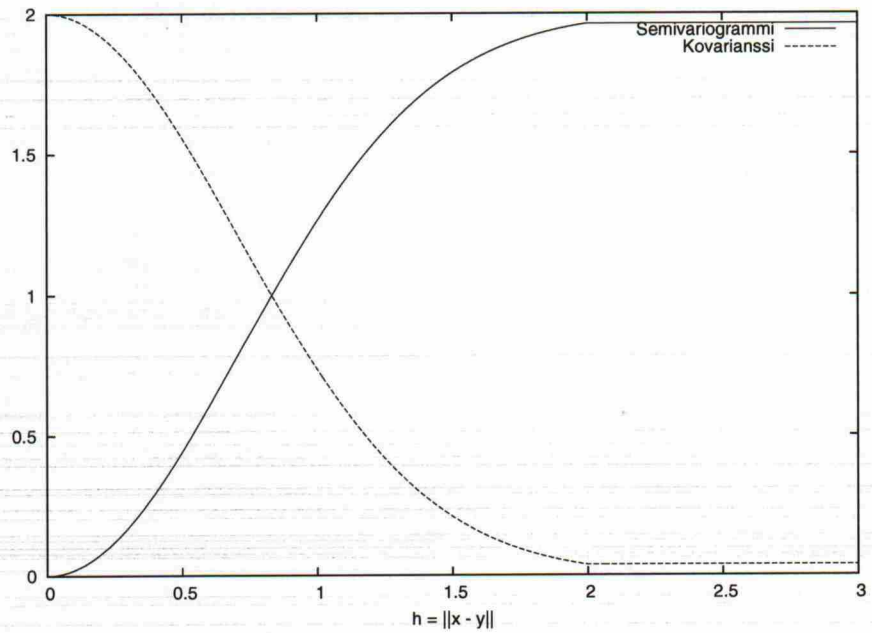
$$p(\mathbf{z}_1 | \, \mathbf{z}_2) \sim \mathcal{N}(\mu_1 + \mathbf{\Sigma_{12}}\mathbf{\Sigma_{22}^{-1}}(\mathbf{z}_2 - \mu_2), \mathbf{\Sigma_{11}} - \mathbf{\Sigma_{12}}\mathbf{\Sigma_{22}^{-1}}\mathbf{\Sigma_{21}}).$$

Tämä tulos on annettu esimerkiksi lähteessä [Gel 03]. Neliöllistä virhettä minimoivat kriging-interpolaatiomenetelmät sivuavat tätä yleisempää tulosta ja ne antavat samoja tai samantyyppisiä vastauksia interpolaatio ongelmaan. Yksinkertainen kriging olettaa keskiarvon tunnetuksi vakioksi ja käyttää tuloksen mukaista keskiarvoa suoraan valitsemalla $\mu = \mu\mathbf{1}$. Tavallisessa krigingissä sen sijaan minimoidaan neliöllinen virhe keskiarvon μ ollessa tuntematon, jolloin ei myöskään oteta kantaa μ:n jakaumaan. Universaalikrigingissä minimoidaan estimoitu neliöllinen virhe olettamalla keskiarvon μ olevan paikanmukainen funktio ja ratkaisemalla samalla tämän funktion kertoimet $\{w_l\}$, jolloin keskiarvon estimaatti on muotoa

$$\mu(\mathbf{x}) = \sum_{l=1}^{n} w_l f_l(\mathbf{x}).$$

Koska universal kriging johdetaan hyvin eri tavalla kuin ehdollisen normaalijakauman ehdollinen keskiarvo, jää näiden kahden suhde ilman tarkempaa tarkastelua epämääräiseksi.

Toinen geostatistiikan oma käsite on *semivarianssin* $\gamma(\mathbf{h})$ käyttö, joka on kovarianssille vaihtoehtoinen tapa esittää pisteiden välinen korrelaatio:



Kuva 2.1: Semivariogrammin ja kovariogrammin suhde

$$\gamma(\mathbf{h}) = \tfrac{1}{2} Var[\mathbf{z}(\mathbf{x}) - \mathbf{z}(\mathbf{x}+\mathbf{h})]$$

Kovarianssin ja semivarianssin välinen yhteys on helpompi nähdä, kun oletetaan $\mathbf{z}(\mathbf{x})$:n olevan aina toisen asteen statiikkoihin asti paikan suhteen stationäärinen. Tällöin semivarianssi voidaan esittää muodossa

$$\gamma(\mathbf{h}) = Cov(\mathbf{0}) - Cov(\mathbf{h}).$$

Semivarianssin kuvaajaa etäisyyden funktiona kutsutaan semivariogrammiksi. Edellisen kaavan mukaista semivariogrammin ja kovariogrammin suhdetta on havainnollistettu kuvassa 2.1.

Pelkän määritelmällisen eron lisäksi semivarianssilla on muutamia muita etuja kovarianssin käyttöön verrattuna. Semivarianssin etuja kovarianssiin nähden on sen teoreettisesti laajempi olemassa olo ja semivarianssiestimaattorin parempi tarkkuus suhteessa kovarianssiestimaattorin tarkkuuteen [Ole 99]. Esimerkiksi diskreettillä Wiener-Levy prosessilla ei ole olemassa kovarianssia, mutta sille voidaan laskea semivarianssi.

2.2.2 Valuma-alueiden todennäköisyysalueet

Valuma-alueiden laskemiseksi epätarkoista korkeusmalleista mallinnetaan korkeusmallien korkeusmatriisin epävarmaa virhepintaa jakaumana. Jakaumamallinnusta käytetään, sillä maanpinnan korkeuksien mittaukset ovat käytännössä aina enemmän tai vähemmän epätarkkoja ja niitä voidaan tehdä vain rajallinen määrä. Tämän takia korkeusmatriisin keskiarvo ei koskaan konvergoidu

oikeaan arvoonsa vaan lopputuloksena on epävarmuutta sisältävä korkeusmalli maanpinnan korkeudesta. Tästä korkeusmallin epävarmuudesta seuraa myös epävarmuutta valuma-alueiden laskennan tuloksiin.

Työssä toteutettavia valuma-alueiden laskentatapaa ja korkeusmallien jakaumia on tutkittu Geodeettisella laitoksella [Oks 05, Oks 6]. Normaalijakauman tai normaalijakaumista muodostetun mikstuurimallin havaittiin olevan teorian mukaisesti realistinen korkeusmalli tutkimuksissa käytetyille digitaalisille, diskretoiduille korkeusmalleille. Normaalijakaumien korrelaation havaittiin ja vahvistettiin - osittain jo aiemman tutkimuksen mukaisesti - muuttuvan vahvasti etäisyyden funktiona, minkä seuraksena tutkimuksissa on käytetty etäisyyden funktiona muuttuvia semivariogrammeja $\gamma(\mathbf{h})$ [Ker 00]. Korkeusmallitutkimuksissa semivariogrammia $\gamma(\mathbf{h})$ on parametrisoitu eksponentiaalisesti ja gaussisesti (Kuva 2.2):

$$\gamma_{Exp}(\mathbf{h}) = \sigma^2(1 - e^{-\|\mathbf{h}\|/\phi})$$

$$\gamma_{Gau}(\mathbf{h}) = \sigma^2(1 - e^{-\|\mathbf{h}\|^2/\phi^2}).$$

Korkeusmallista $p(\mathbf{X})$ lasketun valuma-alueen $\mathbf{f_y}(\mathbf{X})$ laskennan tuloksena voitaisiin teoriassa laskea valuma-alueiden jakauma, jonka eksakti analyyttinen muoto olisi luultavasti hyvin monimutkainen ja vaikeasti hahmotettava. Tämän takia on mielekkäämpää ja myöskin helpompaa laskea varsinaisen jakauman sijaan vain sen tärkeimmät tunnusluvut eli keskiarvo

$$E[\mathbf{f_y}(\mathbf{X})] = \int \mathbf{f_y}(\mathbf{X})p(\mathbf{X})d\mathbf{X}$$

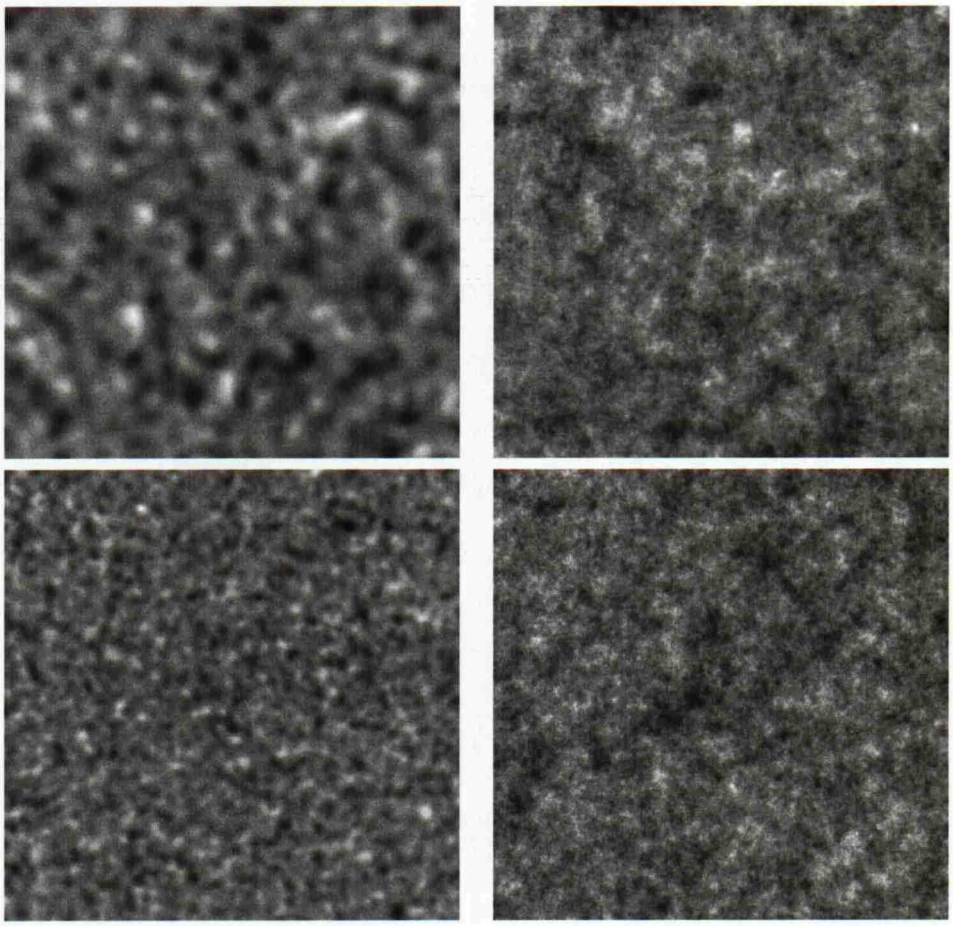
ja varianssit

$$Var[\mathbf{Y}(i,j)] = \int(\mathbf{Y}(i,j)^2 - E[\mathbf{Y}(i,j)]^2)p(\mathbf{X})d\mathbf{X} \ , \ \mathbf{Y} = \mathbf{f_y}(\mathbf{X}).$$

Tunnuslukujen laskemisen ongelmaksi tulee integrointi monimutkaisen, epäjatkuvan korkeusmatriisin valuma-alueita laskevan funktion $\mathbf{f_y}$ yli, jonka suora laskeminen on erittäin vaikeaa tai liki mahdotonta. Tämän takia odotusarvon integraaleja approksimoitiin seuraavaksi esiteltävällä Monte Carlo integroinnilla eli simuloinnilla.

2.2.3 Monte Carlo integrointi ja valuma-alueiden laskenta

Monte Carlo eli MC-menetelmiksi kutsutaan satunnaisuutta hyödyntäviä laskentamenetelmiä, jotka antavat aina jonkin vastauksen ongelmaan, mutta annettu vastaus ei välttämättä ole oikea tai erityisen tarkka. Monte Carlo algoritmi voi antaa oikean vastauksen esimerkiksi vain jollakin todennäköisyydellä p. Vastaparina Monte Carlo menetelmille ovat ns. Las Vegas satunnaislaskenta menetelmät, jotka palauttavat aina oikea vastauksen, mutta niiden käyttämä laskenta-aika on satunnainen tai ääretön. Koska MC-menetelmät antavat Las Vegas algoritmejä heikomman vastauksen ongelmaan, ovat ne yleensä Las



Kuva 2.2: Gaussisesti (vas.) ja eksponentiaalisesti (oik.) korreloituneet, normaalijakautuneet virhepinnat ($\phi = \{8, 4\}, \sigma = 1$)

Vegas algoritmejä nopeampia. MC-menetelmiä käytetään mm. integroinnissa, optimoinnissa sekä satunnaisprosessien ja -muuttujien näytteistyksessä.

Analysoitaessa korkeusmallien valuma-alueiden epävarmuutta käytetään Monte Carlo- menetelmiä arvioimaan odotusarvon integraalia

$$E[\mathbf{f_y}(\mathbf{X})] = \int \mathbf{f_y}(\mathbf{X})p(\mathbf{X})d\mathbf{X},$$

missä $p(\mathbf{X})$ on jakauma korkeusmallin mahdollisille arvoille ja $f(\mathbf{X})$ on 0/1-arvoinen epäjatkuva valuma-alueanalyysifunktio. Aiemmin kuvattujen tutkimustulosten perusteella on havaittu, että kohtuullinen approksimaatio jakaumalle $p(\mathbf{X})$ on normaalijakauma. Integraalia voidaan siten arvioida luomalla korkeusmallin realisaatioita normaalijakaumasta $p(\mathbf{X})$ ja laskemalla keskiarvo

$$\mathbf{M}_N = \frac{1}{N} \sum \mathbf{f_y}(\mathbf{X}).$$

Tämä on julkaisujen [Oks 05] ja [Oks 6] mukainen tapa laskea valuma-alueiden todennäköisyyskarttoja epätarkkuutta sisältävistä korkeusmalleista. Julkaisujen mukainen eksaksti laskentatapa laskea valuma-alueet on:

1. Luodaan normaalijakautunut $\mathcal{N}(vec(\mathbf{X}_0), \mathbf{\Sigma}(\sigma^2, \phi))$ virhepinta, missä $\mathbf{X}_0$ on annettu korkeusmallin korkeusmatriisi ja $\mathbf{\Sigma}(\sigma^2, \phi)$ on valitun semivariogrammin parametrien mukainen spatiaalinen korrelaatio.

2. Täytetään virhettä sisältävän korkeusmallin kuopat.

3. Painetaan virtausuomastot virhettä sisältävään karttaan, jotta tiedettyjen jokien tai purojen virtaukset eivät muutu. Näin varmuudella tiedettyjen jokien virtaukset eivät pääse katkeamaan.

4. Lasketaan pisteet, jotka voivat virrata annettuihin valumapisteisiin. Asetetaan nämä kartan pisteet ykkösiksi ja muut nolliksi.

5. Käytetään laskettua valuma-aluetta $\mathbf{D}_i$ laskemaan uusi tarkempi keskiarvo $\mathbf{P}_i = \frac{1}{N} \sum \mathbf{D}_i = \frac{1}{N}[\mathbf{D}_i + (N-1)\mathbf{P}_{i-1}]$.

Tätä laskentatapaa käytettiin myös toteutetussa ohjelmassa virtausuomaston painamista lukuunottamatta. Toteutetun ohjelman laskentamenetelmä on kuvattu tarkasti luvussa viisi.

Menetelmällä saadaan laskettua kartan valuma-alueet, mutta se ei arvioi Monte Carlo simuloinnilla lasketulle keskiarvon estimaattorille mitään tarkkuusarviota. Tulosten luotettavuuden kannalta tämä olisi kuitenkin tärkeää. Keskiarvo estimaatin virhettä voidaan arvioida esimerkiksi ns. monitorointimenetelmällä.

2.2.4 Konvergenssin monitorointi

Keskiarvon laskemisessa käytetty Monte Carlo-simulointimenetelmä konvergoituu varmasti oikeaan tulokseensa vasta, kun korkeusmallien realisaatioiden määrä kasvaa rajatta. Tämän takia lasketun keskiarvon jäljellä olevalle virheelle pitäisi pyrkiä laskemaan yläraja tavalla, jonka laskeminen on mahdollista myös hyvin suurilla korkeusmalleilla.

Yhdelle muuttujalle soveltuu keskiarvon konvergenssin arviointimenetelmäksi menetelmä, jota lähteessä [Rob 04] kutsutaan konvergenssin monitorointimenetelmäksi. Se olettaa muuttujien olevien normaalijakautuneita, mutta antaa kohtuullisia tuloksia myös muissakin tapauksissa. Tämä ei ole erityisen yllättävää, koska keskiarvon jakauman tiedetään lähestyvän normaalijakaumaa näytteiden määrän kasvaessa suureksi, lisäksi maksimientropiajakaumana normaalijaukama on informaatioteoreettisessa mielessä maksimaalisen satunnainen. Tässä kappaleessa kuvataan kirjassa annettu menetelmä, joka soveltuu yhdelle muuttujalle. Menetelmän laajennus useampiulotteisille muuttujille, kuten korkeusmalleille, johdetaan luvussa kolme.

Merkitään havaintosarjaa riippumattomilla muuttujilla $x_1, x_2, ..x_K$, jolloin harhaton keskiarvon estimaattori hetkellä $i = K$ on

$$m_K = \frac{1}{K} \sum x_i,\ i = 1...K.$$

Tämän perusteella voidaan keskiarvon estimaattoreille m_K ja m_L laskea kovarianssi

$$Cov(m_K, m_L) = E[(m_K - \mu_x)(m_L - \mu_x)^T].$$

Valitsemalla nyt keskiarvoksi nolla tuloksen yleisyyden siitä kärsimättä nähdään kovarianssin olevan

$$Cov(m_L, m_K) = E[\frac{1}{KL} \sum x_i x_j] = \frac{\sigma_x^2}{max(K,L)},$$

koska muuttujat $\{x_i\}$ ovat keskenään riippumattomia. Tuloksen perusteella voidaan prosessin keskiarvomuuttujalla $\mathbf{m}_K = [m_1 m_2 .. m_K]^T$ nähdä olevan jakauma $\mathbf{m}_K \sim \mathcal{N}_K(\mathbf{1}\mu_x, \mathbf{\Sigma}_{\mathbf{m}_K})$, missä kovarianssimatriisi on muotoa

$$\mathbf{\Sigma}_K = \sigma_x^2 \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots & \frac{1}{K} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots & \frac{1}{K} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \cdots & \frac{1}{K} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{1}{K} & \frac{1}{K} & \frac{1}{K} & \frac{1}{K} & \frac{1}{K} & \cdots & \frac{1}{K} \end{bmatrix}.$$

Matriisi voidaan kääntää algebrallisesti, jolloin tulokseksi saadaan tridiagonaalinen matriisi

$$\mathbf{\Sigma}_K^{-1} = \frac{1}{\sigma_x^2} \begin{bmatrix} 2 & -2 & 0 & 0 & 0 & \cdots & 0 \\ -2 & 8 & -6 & 0 & 0 & \cdots & 0 \\ 0 & -6 & 18 & -12 & 0 & \cdots & 0 \\ 0 & 0 & -12 & 32 & -20 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & -K(K-1) \\ 0 & 0 & 0 & 0 & \cdots & -K(K-1) & 2K^2 \end{bmatrix}.$$

Nyt jos varianssia σ_x^2 ei tunneta, sen estimaatti $\hat{\sigma}_x^2$ joudutaan laskemaan mittauksista ja

konvergenssin kannalta oleellinen termi noudattaa jakaumaa

$$(\mathbf{m}_K - \mathbf{1}\mu)^T \mathbf{\Sigma}_K^{-1} (\mathbf{m}_K - \mathbf{1}\mu) \sim n F_{n,\nu},$$

$$\text{missä } n = K \text{ ja } \hat{\sigma}_x^2 \sim \chi_\nu^2 \, .$$

Oleellista yllä on, että varianssiestimaatori $\hat{\sigma}_x^2$ on riippumaton keskiarvon estimaattorista. Mikäli varianssi taas tunnetaan, on normaalijakauman eksponentiaalisen termin jakauma Ksiin neliön jakauma χ_K^2. Tulosten perusteella keskiarvolle μ voidaan siten ratkaista luottamusväli. Jakauman $f_x(x)$ kertymäfunktiosta $F_x(x)$ ratkaistaan luottamusväli eksponentiaaliselle termille $d = F^{-1}(0.95)$, josta voidaan laskea luottamusväli myös keskiarvolle:

$$\{\mu : (\mathbf{m_K} - \mathbf{1}\mu)^{\mathbf{T}} \mathbf{\Sigma_K^{-1}} (\mathbf{m_K} - \mathbf{1}\mu) \leq d\} \Rightarrow$$

$$\mu^2 \mathbf{1}^{\mathbf{T}} \mathbf{\Sigma_K}^{-1} \mathbf{1} - 2\mu \mathbf{m_K}^T \mathbf{\Sigma}_K^{-1} \mathbf{1} + (\mathbf{m_K^T \Sigma_K}^{-1} \mathbf{m_K} - d) \leq 0$$

$$\mu^2 (K^2 + K)\sigma_x^{-2} - 2m_K (K^2 + K)\sigma_x^{-2} \mu + \mathbf{m_K^T \Sigma_K}^{-1} \mathbf{m_K} - d \leq 0$$

$$\mid \mu - m_K \mid \leq \sqrt{m_K^2 + \tfrac{\sigma_x^2}{K(K+1)} \left(d - \mathbf{m_K^T \Sigma_K^{-1} m_K}\right)}.$$

Näin saadaan laskettua eksakti 95% luottamusväli keskiarvoestimaatin virheelle.

2.3 Satunnaismuuttujien simulointi

Monte Carlo simulaatiossa odotusarvoa arvioidaan käyttämällä odotusarvon jakauman mukaan jakaantuneita satunnaismuuttujia keskiarvon laskemiseen. Valuma-alue analyysissä jakaumana käytetään korkeaulotteista, spatiaalisesti korreloitunutta normaalijakaumaa. Luvussa esitellään aluksi riippumattomien normaalijakautuneiden muuttujien generointimenetelmiä, jonka jälkeen paikalliset korrelaatiot lisätään virhepintaan käyttämällä prosessikonvoluutiomenetelmää.

Teoriassa korreloimattomien normaalijakautuneiden satunnaismuuttujien generointi on helppoa. Keskeisen raja-arvolauseen mukaan summa miten tahansa jakautuneita satunnaismuuttujia konvergoituu kohti normaalijakaumaa,

$$\epsilon = \tfrac{1}{N} \textstyle\sum \epsilon_i, \; \epsilon \sim N(\mu, \sigma^2).$$

Tämä ei kuitenkaan ole käytännöllinen, nopea tai luotettava tapa generoida normaalijakautuneita satunnaislukuja. Tilastollisen erityisasemansa ja käytännön tarpeen takia tehokkaita normaalijakautuneiden muuttujien generointimenetelmiä on tutkittu ja kehitetty paljon.

2.3.1 Normaalijakautuneet satunnaismuuttujat

Riippumattomien normaalijakautuneiden pseudosatunnaismuuttujien luomisen hyvänä perusmenetelmänä on Box-Mullerin menetelmä [Box 58]. Menetelmä perustuu lähes kaikkien ohjelmointikielien tukemien tasajakaantuneiden $p_{\mathbf{x}}(\mathbf{x}) = 1$ muuttujien epälineaariseen muutokseen $\mathbf{y} = \mathbf{f}(\mathbf{x})$, jolloin saatu jakauma on muotoa

$$p_{\mathbf{y}}(\mathbf{y}) = p_{\mathbf{x}}(\mathbf{x}) \left| \frac{\partial \mathbf{x}}{\partial \mathbf{y}} \right|.$$

Box-Mullerin menetelmässä luodaan satunnaismuuttujat pareittain tasajakaantuneista satunnaismuuttujista käyttämällä muunnoksia [Pre 92]

$$y_1 = \sqrt{-2 \ln x_1} \cos 2\pi x_2,$$
$$y_2 = \sqrt{-2 \ln x_1} \sin 2\pi x_2.$$

Funktio voidaan kääntää muotoon

$$x_1 = \exp\left[-\tfrac{1}{2}(y_1^2 + y_2^2)\right],$$
$$x_2 = \tfrac{1}{2\pi} \arctan \tfrac{y_2}{y_1}.$$

Tällöin Jakobin determinantiksi saadaan

$$\frac{\partial(x_1,x_2)}{\partial(y_1,y_2)} = \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} = -\left[\tfrac{1}{\sqrt{2\pi}} e^{-y_1^2/2}\right]\left[\tfrac{1}{\sqrt{2\pi}} e^{-y_2^2/2}\right]$$

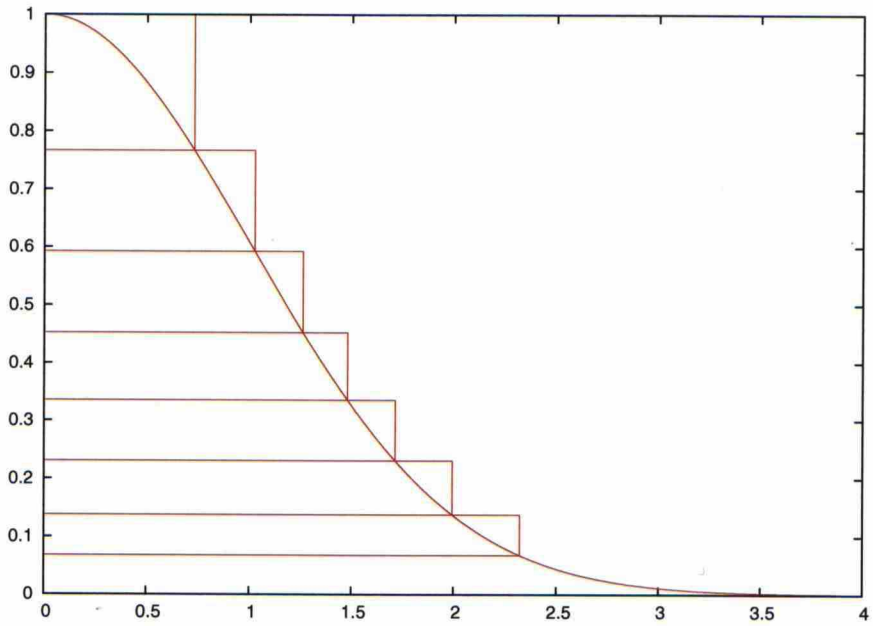
eli kahdesta tasajakaantuneesta muuttujasta saadaan kaksi normaalijakaantunutta, yksikkövarianssista muuttujaa

$$p_{y_1 y_2}(y_1, y_2) = \left|\frac{\partial(x_1,x_2)}{\partial(y_1,y_2)}\right| = \tfrac{1}{\sqrt{2\pi}} e^{-y_1^2/2} \tfrac{1}{\sqrt{2\pi}} e^{-y_2^2/2}.$$

Menetelmä on yksinkertainen ymmärtää ja toteuttaa, mutta se ei kuitenkaan ole erityisen nopea menetelmä suurien muuttujamäärien luomiseksi. Vaikka sinin ja kosinin [Pre 92] laskeminen voidaan eliminoida menetelmää jatkokehittämällä, on logaritmien laskeminen silti yksi hitaimpia PC:n laskemia matemaattisia perusfunktioita.

Huomattavasti yksinkertaisempi ja nopeampi tapa luoda riippumattomia normaalijakautuineita satunnaismuuttujia on approksimoida normaalijakaumaa riittävän tarkasti suorakaiteen muotoisilla alueilla ja käyttää sen jälkeen hylkäysnäytteistystä (rejection sampling). Marsagalian Ziggurat menetelmä [Mar 00] perustuu tähän ideaan. Menetelmällä saadaan luotua 400 MHz:n PC koneessa n. 15 miljoonaa muuttujaa sekunnissa eli n. 15 kappaletta 1000×1000-kokoista matriisia/s ja likimain yksi 4000×4000 matriisi/s.

Hylkäysnäytteistyksessä luodaan satunnaisluku jakaumasta $f(\mathbf{x})$ määrittelemällä alue A, jonka sisään alue $B = \{(\mathbf{x}, y) \mid y < f(\mathbf{x})\}$, $B \subset A$, jää. Tämän jälkeen alueesta A generoidaan tasajakaantuneita vektoreita $\mathbf{z}$ kunnes $\mathbf{z} \in B$, joka hyväksytään näytteeksi jakaumasta $f(\mathbf{z})$. Näin luodut näytteet ovat jakauman $f(\mathbf{z})$ mukaisesti jakautuneita. Ziggurat-menetelmässä hylkäysnäytteistystä käytetään hyväksi peittämällä, jakauman häntää lukuunottamatta, positiivinen osa normaalijakaumausta yhtäsuuren kokoisilla suorakaiteilla A_i. Suorakaiteiden koko on lisäksi valittu siten, että niiden pinta-ala on yhtä suuri häntäosan kanssa (Kuva 2.3). Kun alueiden jakokohdat $\{x_i\}$ x-akselilla on ratkaistu, ovat suorakaiteen muotoiset alueet määriteltävissä seuraavasti:



Kuva 2.3: Ziggurat menetelmä normaalijakaumalle, $N = 8$

$$A_i = \{(x,y) \mid y < f(x_i), x \in [x_i, x_{i+1}[\}, \; i \neq N,$$
$$A_N = \{(x,y) \mid y < f(x), x \geq x_N\} \text{ ja}$$
$$A = \bigcup A_i \; , \; 0 = x_0 < x_1 < x_2 < ... < x_N.$$

Näin suorakaiteista ja häntäosasta on helppo luoda tasajakautuneita muuttujia valitsemalla aina satunnaisesti jokin niistä ja yrittämällä sen jälkeen luoda hylkäisnäytteistyksellä näyte ko. normaalijakauman alueesta. Häntäosasta satunnaismuuttuja luodaan suoraan hitaammalla menetelmällä. Käytännön nopeuden kannalta merkittävää on lisäksi se, että menetelmä voidaan saada toteutettua häntää lukuunottamatta kokonaisluvuilla ja muutamalla etukäteen lasketulla taulukolla.

2.3.2 Prosessikonvoluutio

Prosessikonvoluutio on laskennallisesti nopea tapa lisätä halutunlainen, spatiaalinen autokorrelaatio tason tai avaruuden pisteille. Teoriassa haluttu korrelaatio riippumattomiin muuttujiin ($\mathbf{\Sigma_x} = E[(\mathbf{x}-\mu_\mathbf{x})(\mathbf{x}-\mu_\mathbf{x})^T] = \mathbf{I}$) voidaan saada aikaan helposti käyttämällä hyväksi kovarianssimatriisin ominaisarvohajotelmaa $\mathbf{\Sigma} = \mathbf{X\Lambda X^T}$. Sisältäköön kovarianssimatriisi $\mathbf{\Sigma_y}$ halutun tyyppisen korrelaation $\mathbf{y}$:n elementtien välillä. Satunnaismuuttuja $\mathbf{y}$:n mukaisesti korreloituneita vektoreita voidaan nyt luoda korreloimattomista satunnaismuuttujista $\mathbf{x}$ laskemalla $\mathbf{z} = \mathbf{Xx}$. Tämä muuttujien vaihto ei myöskään vaikuta odotusarvon jakaumaan, koska symmetrisellä kovarianssimatriisilla ominaisarvohajotelman $\mathbf{X}$ matriisit ovat rotaatiomatriiseja, joiden determinantti on yksi [Gol 83]. Näin luotujen satunnaismuuttujien korrelaatio on siten

$$\begin{aligned}\Sigma_{\mathbf{z}} &= E_{\mathbf{z}}[(\mathbf{z}-\mu_{\mathbf{z}})(\mathbf{z}-\mu_{\mathbf{z}})^T] \\ &= E_{\mathbf{x}}[(\mathbf{X}\mathbf{x}-\mathbf{X}\mu_{\mathbf{z}})(\mathbf{X}\mathbf{z}-\mathbf{X}\mu_{\mathbf{z}})^T] \\ &= \mathbf{X}E_{\mathbf{x}}[(\mathbf{x}-\mu_{\mathbf{z}})(\mathbf{z}-\mu_{\mathbf{z}})^T]\mathbf{X}^T \\ &= \mathbf{X}\Sigma_{\mathbf{x}}\mathbf{X}^T = \mathbf{X}\mathbf{X}^T = \Sigma_{\mathbf{y}}.\end{aligned}$$

Tämä on kuitenkin vain teoreettinen ratkaisu, koska käytännössä ongelmaksi tulee aiemmin mainittu kovarianssimatriisin muistin käytön liian nopea $O(N^4)$ kasvu. $10^3 \times 10^3$-kokoiselle korkeusmallilla ($N = 10^3$) kovarianssimatriisin koko olisi jo miltei 10^{12} lukuarvoa eli 512 GB.

Prosessikonvoluutiossa [Swa 99, Ker 00] käytetään hyväksi signaalinkäsittelyn konvoluutioteoreemaa [Gon 99] halutunlaisen korrelaation laskemiseksi muuttujien välille. Menetelmän ainoana rajoituksena on luotujen korrelaatioiden paikallisuus. Suodattamalla nollakeskiarvoista ja yksikkövarianssista normaalikohinaa voidaan prosessikonvoluutiolla luoda $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_{\mathbf{x}}(\sigma^2, \phi))$ jakautuneita muuttujia, missä korrelaatiomatriisi on matriisin muotoon asetettujen satunnaismuuttujien semivarianssin $\gamma(\mathbf{h})$ parametrien σ^2, ϕ mukaisesti laskettavissa oleva korrelaatiomatriisi $\mathbf{\Sigma}_{\mathbf{x}}(i,j) = Cov(\mathbf{x}_i - \mathbf{x}_j)$, missä $\mathbf{x}_i$ ja $\mathbf{x}_j$ ovat matriisiin asetettujen satunnaismuuttjien i ja j koordinaatit. Menetelmässä tarvittava konvoluutio voidaan laskea nopeasti Fourier muunnosta hyödyntävällä FFT eli Fast Fourier Transform algoritmin avulla [Mit 98].

Rajoittamalla korkeusmallin pisteiden välisen etäisyyden mukainen autokorrelaatio $\rho(\mathbf{h}) = Cov(\mathbf{h})$ ja siten myös semivarianssi $\gamma(\mathbf{h})$ samaksi kaikille korkeusmallin pisteille, voidaan korrelaatioiden lisääminen muuttujien välille toteuttaa kaksiulotteisena diskreettinä konvoluutiona $\mathbf{M} \star \mathbf{N}$. Olkoon $\mathbf{M}$ ja $\mathbf{N}$ kaksi kaksiulotteista funktiota tai matriisia
$\mathbf{M} : [A_0, A_1] \times [B_0, B_1] \longrightarrow \Re$, $\mathbf{N} : [X_0, X_1] \times [Y_0, Y_1] \longrightarrow \Re$. Diskreettikonvoluutio funktioiden välillä on

$$(\mathbf{M} \star \mathbf{N})[x,y] = \sum_{a=A_0}^{A_1} \sum_{b=B_0}^{B_1} \tilde{\mathbf{M}}[a,b]\tilde{\mathbf{N}}[x-a, y-b].$$

Kaavassa $\tilde{\mathbf{M}}$ ja $\tilde{\mathbf{N}}$ ovat funktioiden $\mathbf{M}$ ja $\mathbf{N}$ laajennoksia kaikille kokonaisluvuille

$$\tilde{\mathbf{M}}[x,y] = \begin{cases} \mathbf{M}[x,y] & , x \in [A_0, A_1], y \in [B_0, B_1] \\ 0 & , \textit{muulloin} \end{cases}.$$

Prosessikonvoluutiossa funktioista $\mathbf{M}$ ja $\mathbf{N}$ toinen funktioista on korkeusmallin muotoinen, riippumatonta normaalijakautunutta kohinaa sisältävä matriisi ja toinen on semivarianssin mukaan laskettu matriisi eli konvoluutio- tai suodinmaski. Mikäli semivarianssin mukainen korrelaatio on paikallista eli etäisyyden mukaiselle korrelaatiokertoimelle $\rho(\mathbf{h})$ pätee

$$\|\mathbf{h}\| \geq L \Rightarrow \rho(\mathbf{h}) \approx 0,$$

on konvoluutiomaskin koko riippumaton korkeusmallin koosta. Konvoluutiomaskin koko on $(2L+1)^2$ ja korrelaatiokertoimia vastaavan konvoluutiomaskin ei siten tarvitse olla korkeusmallin kokoinen.

2.3.3 Prosessikonvoluution teoriaa

Suodinmaskin laskeminen korrelogrammista

Suodinmaskin laskeminen vaatii konvoluutioteoreeman hyödyntämistä kaksiulotteisessa tapauksessa. Konvoluutioteorian mukaan 2d-kuvien konvoluutio $\mathbf{M} \star \mathbf{N}$ voidaan esittää taajuustasossa Fourier-muunnettujen $\mathbf{F}[u,v] = \mathcal{F}(\mathbf{f}[x,y])$ funktioiden kertolaskuna. Näin voidaan saavuttaa selvä nopeutus konvoluution laskennassa. Konvoluution suora laskeminen kaksiulotteisille, likimain samanmittaisille, diskretoiduille signaaleille vaatii $O(N^3)$ operaatioa, kun taas Fourier-muunnoksen avulla, jonka laskemiseen löytyy nopea FFT-algoritmi, on mahdollista laskea kaksiulotteinen konvoluutio ajassa $O(N^2 logN)$. Kaksiulotteinen Fourier- muunnos $\mathcal{F}$ ja sen soveltaminen konvoluution laskemiseen perustuu kaavoihin

$$\mathcal{F}(\mathbf{M})[u,v] = \mathbf{M}[u,v] \propto \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \mathbf{M}[x,y] e^{-j2\pi(ux/M+vy/N)},$$

$$\mathbf{M} \star \mathbf{N} \propto \mathcal{F}^{-1}\left[\mathcal{F}(\mathbf{M})\mathcal{F}(\mathbf{N})\right].$$

Konvoluutioteoreeman tarkempi todistus on löydettävissä työn liite-osasta. Varsinaisen suodinmaskin laskemisprosessin johtaminen vaatii prosessikonvoluutioteorian eksaktia esittelemistä. Generoitava diskreetti virhepinta lasketaan suodattamalla matriisin $\mathbf{N}$ normaalijakautunutta korreloimatonta kohinaa $\mathbf{N}[n,m] \sim \mathcal{N}(0,1)$ siten että luodun pinnan normaalijakautuneiden pisteiden välille $\mathbf{Z} = \mathbf{M} \star \mathbf{N}$ muodostuu halutunlainen korrelaatio.

Käyttämällä kaksiulotteista diskreetin korrelogrammin määritelmää

$$\mathbf{R_{XY}}[h_1,h_2] \propto E[\mathbf{X}[n,m]\mathbf{Y}[n+h_1,m+h_2]]$$

voidaan laskea kaava autokorrelaatiomatriisin $\mathbf{R_{ZZ}}(\mathbf{M})$ laskemiseksi suodinmaskista $\mathbf{M}$. Rajoittumalla symmetrisiin maskeihin voidaan kaava myös kääntää ja laskea suodinmaski $\mathbf{M}$, kun autokorrelaatiomatriisi $\mathbf{R_{ZZ}} = \mathbf{A}$ on tiedossa.

$$\mathbf{M} = \mathbf{R}_{\mathbf{ZZ}}^{-1}(\mathbf{A})$$

Korreloimattoman satunnaispinnan pisteet ovat riippumattomia, joten pätee

$$E[\mathbf{N}[i,j]\mathbf{N}[i+a,j+b]] = \delta(a,b)$$

Käyttämällä tätä tulosta voidaan laskea autokorrelaatio konvoloidulle kohinalle **Z**:

$$\mathbf{R_{ZZ}}[h_1,h_2] \propto E[\mathbf{Z}[n,m]\mathbf{Z}[n+h_1,m+h_2]]$$

$$= E[(\sum_{j_1=-M}^{M}\sum_{i_1=-N}^{N}\mathbf{M}[i_1,j_1]\mathbf{N}[n-i_1,m-j_1])$$

$$(\sum_{j_2=-M}^{M}\sum_{i_2=-N}^{N}\mathbf{M}[i_2,j_2]\mathbf{N}[n+h_1-i_2,m+h_2-j_2])]$$

$$= \sum_{i_1=-N}^{N}\sum_{j_1=-M}^{M}\mathbf{M}[i_1,j_1]\mathbf{M}[i_1+h_1,j_1+h_2].$$

Tästä on mahdotonta tai erittäin hankalaa ratkaista maskia **M** suoraan. Käyttämällä konvoluutioteoreemaa maski saadaan kuitenkin ratkaistua helposti. Jo aiemmin esitetty kaksiulotteinen Fourier-muunnos on

$$\mathbf{F}[u,v] \propto \sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\mathbf{F}[x,y]e^{-j2\pi(ux/M+vy/N)},$$

joten virhepinnan autokorrelaation Fourier-muunnos on

$$\mathbf{R_{ZZ}}[u,v] \propto \sum_{i_1=-N}^{N}\sum_{j_1=-M}^{M}\mathbf{M}[i_1,j_1]e^{+j2\pi(i_1u/(2M+1)+i_2v/(2N+1))}$$

$$\sum_{h_1=-M}^{M}\sum_{h_2=-N}^{N}\mathbf{M}[i_1+h_1,j_1+h_2]e^{-j2\pi(u(i_1+h_1)/(2M+1)+v(i_2+h_2)/(2N+1))}$$

Jos maskille lisäksi pätee seuraava sirkulaarisuusehto, joka on tarpeen, jotta konvoluutio voitaisiin laskea Fourier-muunnoksen kautta:

$$\mathbf{M}[i_1+h_1,i_2+h_2] = \mathbf{M}[(i_1+h_1) \mod (2N+1),(i_2+h_2) \mod (2M+1)],$$

$$h_1 \in [-2N..2N] \ , \ h_2 \in [-2M..2M],$$

yksinkertaistuu autokorrelaation Fourier-muunnos muotoon

$$(\sum_{i_1=-N}^{N} \sum_{j_1=-M}^{M} \mathbf{M}[i_1, j_1] e^{+j2\pi(i_1 u/(2M+1)+i_2 v/(2N+1))}) \mathbf{M}[u, v].$$

Tulosta voidaan yksinkertaa reaalisen suodinmaskin tapauksessa merkisemällä konjugoitua kompleksiarvoista matriisia merkinnällä $\mathbf{M}^*$, jolloin Fourier-muunnetun autokorrelaatiomatriisin arvot yksinkertaistuvat muotoon

$$\mathbf{R_{ZZ}}[u, v] \propto \mathbf{M}^*[u, v]\mathbf{M}[u, v] = \|\mathbf{M}[u, v]\|^2.$$

Kaava voidaan kirjoittaa myös muodossa $\mathbf{R_{ZZ}}[u, v] = \alpha\|\mathbf{M}[u, v]\|^2$,missä α on reaalinen skaalaustermi. Tämä menetelmä autokorrelaation Fourier-muunnoksen laskemiseksi sopii mille tahansa suodinmaskille. Aiemmin mainittu maskin sirkulaarisuusehto voidaan nimittäin aina saattaa voimaan laskujen ajaksi lisäämällä suotimen loppuun riittävä määrä nollia.

Tuloksesta saadaan $\|\mathbf{M}[u, v]\| = \alpha^{1/2}\mathbf{R_{ZZ}}[u, v]^{1/2}$, mikä ei kuitenkaan ratkaise maskia kokonaan. Mikäli suodinmaski on symmetrinen eli $\mathbf{M}[-i, -j] = \mathbf{M}[i, j]$, voidaan suodinmaski kuitenkin ratkaista:

$$\begin{aligned}
\mathbf{M}^*[u, v] &= \textstyle\sum_{i_1=-N}^{N} \sum_{j_1=-M}^{M} \mathbf{M}[i_1, j_1] e^{+j2\pi(i_1 u/(2M+1)+i_2 v/(2N+1))} \\
&= \textstyle\sum_{i_1=-N}^{N} \sum_{j_1=-M}^{M} \mathbf{M}[-i_1, -j_1] e^{+j2\pi(i_1 u/(2M+1)+i_2 v/(2N+1))} \\
&= \textstyle\sum_{i_1=-N}^{N} \sum_{j_1=-M}^{M} \mathbf{M}[i_1, j_1] e^{-j2\pi(i_1 u/(2M+1)+i_2 v/(2N+1))} \\
&= \mathbf{M}[u, v].
\end{aligned}$$

Tällöin $\mathbf{M}[u, v] = \alpha^{1/2}\mathbf{R_{ZZ}}[u, v]^{1/2}$ ja tuloksesta seuraa myös maskin reaalisuus Fourier-muunnoksella lasketussa taajuustasossa,

$$\mathbf{M}[u, v] = r_1 e^{-j\beta} = r_1 e^{j\beta} \Rightarrow \beta = 0.$$

Tämä rajoittaa menetelmälle sopivien reaalisten suodinmaskien $\mathbf{M}$ muotoa vielä lisää. Symmetrinen konvoluutiomaski tulee lähes välttämättä siirtää laskujen kannalta yhteiseen origoon laskujen ajaksi, koska ei-origo keskisen maskin Fourier-muunnos on

$$\mathcal{F}(\mathbf{f}(n - n_0, m - m_0)) = \mathbf{F}(u, v) e^{-j2\pi(un_0/N, vm_0/M)}.$$

Vaihtoehtoisesti voitaisiin laskuissa kertoa Fourier-muunnoksia sopivasti valituilla arvoilla niin että laskut menevät yhä oikein, toteutuksessa on kuitenkin yksinkertaisinta pitää kaikki laskut FFT:n kannalta origo keskitettynä niin, että

origoksi valitaan taulukon alku $[0,0]$ eikä $[\frac{N}{2}, \frac{M}{2}]$. Laskettu autokorrelaatiomatriisi $\mathbf{R_{ZZ}}$ tulee laskea tämän mukaisesti ja autokorrelaatiomatriisi pitää ajatella olevan nollilla laajennetun suodinmaskin sirkulaarisen konvoluution tulokseksi. Tämän takia maskia laskettaessa autokorrelaatiomatriisia tulee laajentaa keskeltä nollilla $(2N+1) \times (2M+1)$ koosta $(4N+1) \times (4M+1)$ kokoon.

Lopuksi ratkaistu $\mathbf{M}[u,v]\alpha^{1/2}$ tulee vielä muuntaa käänteisellä Fourier muunnoksella takaisin spatiaalitasoon $\mathbf{M}[n,m]\alpha^{1/2}$, missä voidaan helposti ratkaista skaalaustermi α. Käyttämällä merkintää $vec(\mathbf{A})$ merkitsemään $N \times M$ kokoisen matriisin arvoja kun ne on asetettu NM-ulotteiseen vektoriin, voidaan konvoluutiokaava kirjoitaa myös vektorimuodossa mille tahansa pisteelle $\mathbf{Z}[n,m]$ seuraavasti:

$$\mathbf{Z}[n,m] = vec(\alpha^{1/2}\mathbf{M})^T\mathbf{n},$$

$$Var\left[\mathbf{Z}[n,m]\right] = E[vec(\mathbf{M})^T\mathbf{n}\mathbf{n}^T vec(\mathbf{M})] - \alpha(vec(\mathbf{M})^T E[\mathbf{n}])^2$$

$$= \alpha vec(\mathbf{M})^T \mathbf{I} vec(\mathbf{M})) - 0 = \alpha \|vec(\mathbf{M})\|^2,$$

missä $\mathbf{n}$ on sopivalla tavalla järjestetty korreloimattoman virhepinnan $\mathbf{N}$ vektorointi. Konvoluutiomaskin $\mathbf{M}$ tuntematon skaalaustermi α voidaan nyt ratkaista, koska virhepinnan $\mathbf{Z}$ semivarianssin varianssiparametri $\sigma_z^2 = Var[\mathbf{Z}[n,m]]$ on tiedossa,

$$\alpha = \frac{\sigma_z^2}{\|vec(\mathbf{M})\|^2} = \frac{\sigma_z^2}{\sum_{n,m} \mathbf{M}[n,m]^2}.$$

Yhdistelemällä nämä osatulokset voidaan suodinmaski laskea etäisyyden mukaan muuttuvan kovarianssin eli kovariogrammin autokorrelaatiomatriisista $\mathbf{R}$, joka on paikallisesti keskittyneiden korrelaatioiden tapauksessa huomattavasti koko konvoloitavaa aluetta pienempi.

Korrelogrammin laskeminen semivariogrammista

Edellisen kappaleen mukainen symmetrinen suodinmaski voidaan laskea autokorrelaatiofunktion perusteella. Käytännössä geostatistiikassa käytetään kuitenkin semivariogrammeja pinnan korrelaatioiden määrittelemiseen. Aikaisemmin esiteltyjen tulosten perusteella halutaan käyttää isotrooppista, joko gaussista tai eksponentiaalista, semivariogrammimallia. Nämä ovat

$$\gamma_{Gau}(\mathbf{h}) = \sigma^2(1 - e^{-\|\mathbf{h}\|^2/\phi^2})$$

$$\gamma_{Exp}(\mathbf{h}) = \sigma^2(1 - e^{-\|\mathbf{h}\|/\phi}).$$

Lisäksi malleihin voidaan haluta lisätä paikallista, riippumatonta pisteittäistä virhettä käyttämällä ns. nugget-semivariogrammia

$$\gamma(\mathbf{h}) = \sigma^2(1 - \delta(\mathbf{h})).$$

Korreloimatonta virhepintaa voitaisiin käyttää pelkällä skaalauksella nugget-semivariogrammin mukaisesti jakautuneena virhepintana, mutta sen mukaista satunnaiskohinaa voidaan myös luoda käsittelemättä nugget-semivariogrammia laskentaa lisää monimutkaistavana erikoistapauksena. Epäjatkuvaa nugget-semivariogrammi mallia voidaan käytännössä approksimoida jatkuvalla gaussisella semivariogrammilla. Kun gaussiselle semivariogrammille valitaan $\phi = 10^{-4}$, saadaan nugget-semivariogrammille approksimaatio

$$\gamma_{Gau}(\mathbf{h}) = \sigma^2(1 - e^{-10^8\|\mathbf{h}\|^2}) \approx \sigma^2(1 - \delta(\mathbf{h})).$$

Tällöin $\gamma(\mathbf{0}) = \sigma^2$ ja diskretoidun semivariogrammin muille pisteille $\|\mathbf{n}\| \geq 1$ pätee

$$\gamma(\mathbf{n}) \geq \sigma^2(1 - e^{-10^8}) > \sigma^2(1 - \frac{1}{1+10^8+10^{16}/2}) > \sigma^2(1 - 10^{-15})$$

eli nugget-semivariogrammia, joka vastaa riippumaton kohinaa, voidaan approksimoida riittävän tarkasti myös gaussisella semivariogrammilla.

Korrelogrammin laskeminen

Toisen asteen nollakeskiarvoisille, stationäärisille jakaumilla semivariogrammi voidaan esittää myös muodossa

$$\gamma(\mathbf{h}) = Cov(\mathbf{0}) - Cov(\mathbf{h}), \text{ missä } Cov(\mathbf{h}) = E[x(\mathbf{u})x(\mathbf{u}+\mathbf{h})].$$

Tarvittava autokorrelaatiofunktio voidaan laskea huomaamalla ja merkitsemällä

$$r(\mathbf{h}) = E_{\mathbf{x}}[x(\mathbf{u})x(\mathbf{u}+\mathbf{h})] = Cov(\mathbf{h}) \ , \ Cov(\mathbf{0}) = \sigma^2.$$

Autokorrelaatiofunktiolla saadaan siis kaavat

$$\begin{aligned} r(\mathbf{h}) &= \sigma^2 - \gamma(\mathbf{h}) \\ r_{Gau}(\mathbf{h}) &= \sigma^2 e^{-\|\mathbf{h}\|^2/\phi^2} \\ r_{Exp}(\mathbf{h}) &= \sigma^2 e^{-\|\mathbf{h}\|/\phi}. \end{aligned}$$

Tämän lisäksi pitää myös suodinmaskin riittävän suuri koko laskea. Yhtälöstä $r(\mathbf{h}) < \epsilon$ saadaan tulokset

$$\|\mathbf{h}_{Gau}\| \geq \phi\sqrt{-\ln(\epsilon\sigma^{-2})},$$

$$\|\mathbf{h}_{Exp}\| \geq -\phi\ln(\epsilon\sigma^{-2}).$$

Jonka jälkeen diskreetit korrelogrammin l. autokorrelaation arvot voidaan laskea $2\|\mathbf{h}_*\| + 1$ kokoiseen matriisiin $\mathbf{R_{ZZ}}[-\|\mathbf{h}_*\|..\|\mathbf{h}_*\|, -\|\mathbf{h}_*\|..\|\mathbf{h}_*\|]$,

$$\mathbf{R_{ZZ}}[i,j] = r_*(i,j).$$

Lisäksi voi olla tarvetta haluta käyttää monen semivariogrammin yhdistelmää virhepinnan korrelaatioiden määrittelemisessä. Virhepinnan luominen tässä tapauksessa vaatii yhtä monen riippumattoman satunnaismuuttujapinnan käyttöä. Jos $\mathbf{Z}(\mathbf{h}) = \sum \mathbf{X}_i(\mathbf{h})$ ja muuttujat ovat riippumattomia, eli

$$E[\mathbf{X}_i(\mathbf{x})\mathbf{X}_j(\mathbf{y})] = Cov(\mathbf{x}-\mathbf{y})\delta(i-j).$$

Autokorrelaatioksi, kovarianssiksi ja siten myös semivariogrammiksi saadaan summattujen muuttujien semivariogrammien summa. Kovariogrammille ja semivariogrammille saadaan tällöin laskettua kaavat

$$Cov(\mathbf{h}) = E[\mathbf{Z}(\mathbf{u})\mathbf{Z}(\mathbf{u}+\mathbf{h})] = \sum E[\mathbf{X}_i(\mathbf{u})\mathbf{X}_j(\mathbf{u}+\mathbf{h})] = \sum r_i(\mathbf{h}),$$

$$\gamma(\mathbf{h}) = Cov(\mathbf{0}) - Cov(\mathbf{h}) = \sum r_i(\mathbf{0}) - r_i(\mathbf{h}) = \sum \gamma_i(\mathbf{h}).$$

2.3.4 Konvoluution laskenta

Prosessikonvoluution laskenta nopeuden kannalta on tärkeää pyrkiä toteuttamaan menetelmän vaatima konvoluutio mahdollisimman nopeasti. Konvoluutio voidaan tehdä Fourier-muunnoksella (FFT) ainakin kolmella eri tavalla, joiden laskennallinen skaalautuvuus on sama $O(n \log n)$. Menetelmien muistinlukutavat eroavat toisistaan kuitenkin niin, että nopeuserot algoritmien välillä voivat uusissa PC-koneissa (2004) olla suuria, jopa 20-kertaisia. Laskentatavat ovat [Mit 98]:

- suora konvoluutioteoreeman käyttö,
- paloittainen "overlap-add" menetelmä ja
- paloittainen "overlap-save" menetelmä.

Näistä ensimmäinen käyttää konvoluutioteoreemaa suoraan laskemalla Fourier muunnoksen kummastakin signaalista ja kertomalla ne taajuustasossa. Kaksi jälkimmäistä toimivat tehokkaasti vain mikäli toinen konvoloitavista signaaleista on toista olennaisesti lyhyempi, jolloin konvoluutioteorian mukainen Fourier muunnos tehdään pienissä, paremmin prosessorin välimuistiin mahtuvissa paloissa. Tämän jälkeen paloittainen konvolointi voidaan korjata tehokkaasti niin, että lopputulokseksi saadaan oikea, koko signaalien välinen konvoluutio.

Suorassa menetelmässä lasketaan konvoluutio laajentamalla konvoloitavat signaalit $\mathbf{x} : [0, N-1] \to \mathbb{R}$ ja $\mathbf{y} : [0, M-1] \to \mathbb{R}$ samanpituisiksi $k \in [N, N+M-1] \Rightarrow \mathbf{x}[k] = 0$, $l \in [M, N+M-1] \Rightarrow \mathbf{y}[l] = 0$, jonka jälkeen ratkaisu saadaan laskemalla $\mathbf{x} \star \mathbf{y} = \mathcal{F}^{-1}[\mathcal{F}[\mathbf{x}]\mathcal{F}[\mathbf{y}]]$ aiemman kappaleen 2.3.3. prosessikonvoluution teoriaa mukaisesti.

Overlap-add menetelmässä konvoluutio $\mathbf{z} = \mathbf{y} \star \mathbf{x}$ lasketaan jakamalla $\mathbf{x}$ k:ksi L:n näytteen pituiseksi signaaliksi $kL = N$.

$$\mathbf{x}[n] = \sum \mathbf{x}_k[n]$$

$$n \in [kL, kL + L - 1] \Rightarrow \mathbf{x}_k[n] = \mathbf{x}[n] \wedge \forall r \neq k \mathbf{x}_r[n] = 0$$

Tällöin konvoluutio voidaan laskea $M + L - 1$-kokoisissa paloissa

$$\mathbf{z} = \sum \mathbf{y} \star \mathbf{x}_k = \sum \mathbf{z}_k,$$

jossa osa edellisen konvoluution tuloksesta joudutaan aina lisäämään seuraavaan, jotta lopputulos olisi oikein.

$$\mathbf{z}[kL + n] = \mathbf{z}_k[n - kL] + \mathbf{z}_{k-1}[n - kL] \ , \ 0 \leq n < L$$

Overlap-save menetelmässä konvoluutio lasketaan jakamalla $\mathbf{x}$ osittain limittäisiin L:n näytteen pituisiin osiin ja käyttämällä vain loppuosa lasketuista konvoluutiosta.

$$\mathbf{x}_k[n] = \mathbf{x}[n + m(L - M)] \ , \ 0 \leq n < L, \ L > M$$

$$\mathbf{z}[n + k(L - M)] = (\mathbf{y} \star \mathbf{x}_k)[n], \ M - 1 \leq n \leq L - 1$$

Kummankin menetelmän edut [Mit 98] suoraan konvoluution laskemiseen ovat samat. Paloittaisilla menetelmillä voidaan välttää huomattavasti lyhyemmän signaalin nollilla laajentaminen pidemmän signaalin pituuden mukaan. Tällöin ei myöskään tarvitse käsitellä koko signaalia kerrallaan, jolloin nopean välimuistin tai varsinaisen muistin rajallisuus ei ole ongelma. Toisin kuin suora konvoluutioteoreeman soveltaminen paloittaisten menetelmät soveltuvat myös äärettömän pituisten signaalien konvolointiin.

2.4 Hajautettu laskenta

Tietokoneiden laskentaa voidaan nopeuttaa joko kehittämällä nopeampia algoritmejä laskentaa varten tai lisäämällä ongelmalle käytössä olevaa laskentakapasiteettia, mikä voi vaatia myös itse algoritmien muuttamista. Niin hajautetussa kuin rinnakkaislaskennassakin laskentatehoa kasvatetaan lisäämällä käytettävissä olevien riippumattomien laskentayksiköiden tai -prosessien määrä. Tällöin täytyy usein myös itse laskenta-algoritmeja muuttaa, jotta toiminta olisi tehokasta hajautetussa tai rinnakkaistetussa laskentaympäristössä.

2.4.1 Hajautettu ja rinnakkaislaskenta

Hajautetun ja rinnakkaislaskennan yhteisenä piirteenä on laskentamenetelmien tarve koordinoida toimintaansa muiden, yksittäisestä laskentayksiköstä eli prosessoreista riippumattomasti toimivien laskentayksiköiden välillä. Koska tiedonsiirto yksiköiden välillä on tavallisesti huomattavasti hitaampaa kuin varsinainen laskenta, on algoritmeissä tärkeää pyrkiä minimoimaan laskentayksiköiden välisessä tiedonsiirrosta olevat viiveet. Algoritmien oikean toiminnan takia on myös usein tarpeellista synkronoida yhteisten muuttujien lukemis- ja kirjoitusoperaatioita, mikä hitautensa takia vaikuttaa usein huomattavasti algoritmien toimintanopeuksiin.

Rinnakkaislaskennassa prosessorit ovat hyvin kiinteässä yhteydessä toisiinsa ja niillä saattaa olla yhteistä muistia, jonka kautta ne voivat välittää tietoja toisilleen lähes viiveettä. Tässä työssä termiä rinnakkaislaskenta käytetään viittaamaan tapaukseen, jossa prosessoreilla on yhteistä muistia ja laskenta tapahtuu rinnakkain ajettavissa säikeissä. Näin määriteltynä rinnakkaislaskennassa on tarpeen synkronoida vain yhteisien muuttujien muuttamisoperaatioita.

Hajautetussa laskennassa sen sijaan jokaisella prosessorilla l. laskentasolmulla on oma muistinsa ja prosessorit keskustelevat hitaasti keskenään jonkin tietoverkon välityksellä. Hajautettua laskentaa ja rinnakkaislaskentaa voidaan myös yhdistää ratkomalla jokaisessa hajautetun laskennnan laskentasolmussa ongelmaa rinnakkaislaskennalla.

Normaali rinnakkaislaskennan tietokoneympäristö on SMP l. symmetrinen moniprosessorikoneympäristö (Symmetric multiprocessing), kun taas hajautetussa laskennassa käytetään tietokoneklustereita. Suuren laskentakapasiteetin klustereissa prosessorien kytkentätekniikka on yleensä käyttötarkoitukseensa erityisesti sovitettua tekniikkaa, kun taas hitaammissa normaalia tietokoneverkkoa käyttävissä ns. Beowulf-klustereissa laskentasolmut voivat olla normaaleja Ethernet-verkolla kytköksissä olevia PC-koneita.

Varsinaisen hajautetun ja rinnakkaislaskennan lisäksi voidaan supertietokoneympäristöissä rinnakkaislaskentaa tehdä laitteistoilla, jotka tukevat vektorilaskennan operaatioiden laskemista tehokkaasti rinnakkain. Rinnakaislaskentaa voidaan myös viedä hajautetumpaan suuntaan ns. NUMA järjestelmissä (Non-Uniform Memory Access), jossa kukin prosessori pystyy käyttämään huomattavasti muita nopeammin jotain osaa muistialueesta. Muita eksoottisempia tapoja käyttää laskentaresursseja tehokkaasti on käyttää laskentaan erikseen suunniteltuja prosessoreja kuten DSP-piirejä tai itseään laskennan aikana muokkaavia FPGA-piirejä (Reconfigurable Computing), joiden konfiguraatio tapahtuu prosessorille integroituja muistielementtejä muuttamalla.

Yleensä yhden organisaation hallinnoimia klusterilaskentaympäristön resursseja voidaan myös yhdistää suuremmiksi GRID-ympäristöiksi, joissa laskentaympäristöt ovat tyypillisistä klustereista poiketen heterogeenisiä. Standardisoituja GRID-laskennan ohjelmia ja protokollia on toteutettu Glubus Alliancen Globus Toolkit:issä, jonka osia käytetään useissa GRID-ympäristöissä kuten pohjoismaisessa NorduGRID:issä tai Yhdysvaltojen TeraGRID:issä.

2.4.2 Tekniikkaa

Hajautetussa laskennassa tarvittavaan laskentasolmujen väliseen tiedonsiirtoon ja synkronointiin on olemassa tätä tarkoitusta varten kehitettyjä standardoituja kirjastoja, joiden algoritmit on optimoitu minimoimaan hajautetun laskennan operaatioihin kuluvaa aikaa.

MPI (Message Passing Interface) [MPI95],[And 99] on yksi yleisesti käytössä oleva hajautetun laskennan standardisoitu rajapinta, joka tukee myös heterogeenisiä eri prosessoriarkkitehtuureita käyttävää hajautettua laskentaa. Rinnakkaislaskentaa varten vastaavia ratkaisuja ovat mm. normaali säikeiden käyttö POSIX threads mukaisen standardoidun kirjastorajapinnan kautta tai SMP-laskentaan tarkoitettu OpenMP rajapinta.

Valuma-aluelaskennan toteutuksessa käytettiin MPI-rajapintaa [LAM] hajautetun laskennan toteuttamiseksi ja POSIX threads säikeitä laskennan rinnakkaistamiseksi hajautetun laskennan laskentasolmuissa. MPI-kirjaston käyttöön

päädyttiin OpenMP rajapinnan sijaan, koska MPI-rajapinnalle löytyy ilmaisia hyviä toteutuksia ja se soveltuu paremmin heterogeenisiin prosessoriarkkitehtuuriympäristöihin. Ohjelmaan lisättiin myös tuki käyttää virtuaalisia, ohjelman sisäisesti simuloimia, säikeitä Pth-kirjaston [PTH] avulla, jolloin samalla ohjelman rakenteella sitä voidaan ajaa myös ympäristöissä, jossa POSIX threads rajapinnan käyttö ei onnistu. Näin pystyttiin mm. kiertämään Tieteellisen laskennan klusterilaskentaympäristössä vastaan tulleet säikeiden luonnin satunnaiseen epäonnistumiseen liittyvät ongelmat. Pth-kirjastoa käytettäessä kadotetaan kuitenkin suurin osa laskennan rinnakkaistamisessa saavutetuista eduista.

Luku 3

Tulosten tarkkuuden arviointi

Monte Carlo-simulaatiosta saadun estimaatin tarkkuuden kannalta on olennaista, että laskettava keskiarvo ja käytettävä näytteistysmenetelmä ovat kummatkin konvergoituneet oikeaan arvoonsa tai jakaumaansa. Näistä jälkimmäinen eli itse normaalijakautuneiden näytteiden luominen voidaan tehdä luotettavasti aiemmin esitellyillä menetelmillä, joten on ainoastaan tarpeellista arvioida laskettavan keskiarvon konvergenssia.

Luvussa johdetaan tilastotieteen avulla arvioita valuma-alueeseen kuulumistodennäköisyysestimaatin tarkkuuden kehittymisestä generoitujen valuma-aluerealisaatioiden määrän kasvaessa. Aluksi tarkastellaan yksittäisen pisteen konvergenssia, jonka jälkeen konvergenssiarviota parannetaan ottamalla huomioon sekä korkeusmatriisin pisteiden keskinäinen että laskettujen keskiarvojen välinen korrelaatio. Korrelaatioiden huomioiminen lisää laskettujen arvioiden realistisuutta, mikä yleensä kasvattaa konvergenssiarvion suuruutta. Nämä korrelaatiot voidaan ottaa huomioon käyttämällä normaalimatriisijakaumaa [Gup 99] ja laajentamalla aiemmin esitetty konvergenssin monitorointimenetelmä usealla muuttujalle [Rob 04]. Menetelmän laskennallisen raskauden takia tarkasta monitorointimenetelmästä muokataan nopeampi, valuma-aluelaskentaan solveltuva likiarvoinen konvergenssin arviointimenetelmä.

3.1 Monte Carlo simulaation konvergenssi

Jotta Monte Carlo simulaatiossa laskettu estimaatti olisi mahdollisimman luotettava, tulisi käytettyjen näytteiden l. korkeusmallin realisaatioiden määrän olla mahdollisimman suuri. Toisaalta laskennan hitauden takia olisi hyvä jos laskenta voitaisiin lopettaa mahdollisimman aikaisessa vaiheessa vaarantamatta kuitenkaan laskentatuloksen tarkkuutta. Laskenta-ajan minimoimisen lisäksi konvergenssiarvioista voidaan usein laskea järkeviä arvioita jäljellä olevalle laskenta-ajalle.

Valuma-alueen laskeminen antaa arvon yksi jokaiselle pisteelle sen valuma-alueeseen kuulumistodennäköisyydellä p ja nollan todennäköisyydellä $1-p$, joten n:n realisaation kertymä $p_n n = \sum x_i$ noudattaa jokaisen korkeusmallin pisteen osalta binomijakaumaa. Binomijakauma $f(k)$ ja sen keskiarvo ja varianssi ovat

$$f(k) = \left(\begin{array}{c} n \\ k \end{array} \right) p^k (1-p)^{n-k},$$

$$E[k] = np,\ Var[k] = np(1-p).$$

Tuloksista saadaan kuulumistodennäköisyyden varianssille arvio:

$$Var[p_n] = Var[k/n] = \frac{p(1-p)}{n} \leq \frac{0.5^2}{n}.$$

Jos yhdelle pisteelle lasketun todennäköisyysarvon varianssi halutaan olevan enintään 10^{-4}, täytyy estimaattien keskihajonta olla 10^{-2} (1%) ja toistokertoja tulee olla $n = 2500$.

Tässä kuitenkin on jäänyt huomioimatta pisteiden keskinäinen ja eri iteraatiokierrosten keskiarvojen välinen korrelaatio, joilla voi olla huomattava vaikutus tarvittavien realisaatioiden määrään [Rob 04]. Tämän takia on tarpeen arvioida konvergenssia tarkemmin. Yllä olevasta yksinkertaisesta tuloksesta voidaan kuitenkin jo laskea likimääräisiä arvioita tarvittaville reaalisaatioiden määrille. Esimerkiksi 10% tarkkuudelle eli keskihajonnalla 10^{-1} iteraatioita tarvitaan vain $n = 25$ (Taulukko 3.1).

Taulukko 3.1: Konvergenssiarvio yhdelle muuttujalle

näytteiden määrä n	$\sim$ keskihajonta σ_{max}
25	10%
100	5%
1000	1,58%

3.2 Korreloinnin huomiointi

Konvergenssin tarkemmaksi analysoimiseksi voidaan pyrkiä esimerkiksi vain arvioimaan kovarianssimatriisin suurinta ominaisarvoa, joka asettaa ylärajan keskiarvon $\mathbf{m}_K = \frac{1}{K}\Sigma\mathbf{x}_i$ virheelle.

Todistetaan aluksi lause $E[tr(\mathbf{x}\mathbf{x}^T)] = tr(E[\mathbf{x}\mathbf{x}^T])$, jota tarvitaan virhearvion laskemiseen. Olkoon nollakeskiarvoisten muuttujien $\mathbf{x}$ kovarianssimatriisi on $\mathbf{\Sigma_x}$ ja ominaisarvohajotelma $\mathbf{\Sigma_x} = \mathbf{X}\mathbf{\Lambda}\mathbf{X}^T$. Tällöin voidaan laskea

$$\begin{aligned}
&E[tr(\mathbf{x}\mathbf{x}^T)] = E[\mathbf{x}^T\mathbf{x}] = E[\mathbf{v}^T\mathbf{v}] \\
&= E[\textstyle\sum_i v_i^2] = \sum_i E[v_i^2] = \sum_i tr(\mathbf{\Sigma_v}) \\
&= \textstyle\sum_i \lambda_i = tr(\Lambda) = tr(\mathbf{\Sigma_x}) \\
&= tr(E[\mathbf{x}\mathbf{x}^T]).
\end{aligned}$$

Laskussa on alussa tehty muuttujanvaihdos $\mathbf{v} = \mathbf{X}^T\mathbf{x}$, jolla poistetaan korrelointi muuttujien väliltä, $\mathbf{\Sigma_v} = E[\mathbf{v}\mathbf{v}^T] = \mathbf{X}^T E[\mathbf{v}\mathbf{v}^T]\mathbf{X} = \mathbf{\Lambda}$. Muuttujan vaihdos ei myöskään muuta odotusarvon arvoa koska $\mathbf{X}$ on vain rotaatiomatriisi, jonka determinantti on yksi. Tulos voidaan yleistää muille kuin nollakeskiarvoisille muuttujille $\mathbf{z}$ laskemalla

$$E[tr(\mathbf{z}\mathbf{z}^T)] = E[tr((\mathbf{z}-\mu_\mathbf{z})(\mathbf{z}-\mu_\mathbf{z})^T)] - tr(\mu_\mathbf{z}\mu_\mathbf{z}^T)$$
$$= tr(E[(\mathbf{z}-\mu_\mathbf{z})(\mathbf{z}-\mu_\mathbf{z})^T]) - tr(\mu_\mathbf{z}\mu_\mathbf{z}^T)$$
$$= tr(E[(\mathbf{z}-\mu_\mathbf{z})(\mathbf{z}-\mu_\mathbf{z})^T - tr(\mu_\mathbf{z}\mu_\mathbf{z}^T)])$$
$$= tr(E[\mathbf{z}\mathbf{z}^T].$$

Lasketaan myös tarpeellinen tulos $\mathbf{\Sigma}_{\mathbf{m}_K} = \frac{1}{K}\mathbf{\Sigma}_\mathbf{x}$,

$$\mathbf{\Sigma}_{\mathbf{m}_K} = E[(\tfrac{1}{K}\textstyle\sum_i \mathbf{x}_i - \mu)(\tfrac{1}{K}\sum_j \mathbf{x}_j - \mu)^T] = \tfrac{1}{K}E[\tfrac{1}{K}\textstyle\sum_i(\mathbf{x}_i-\mu)\sum_j(\mathbf{x}_j-\mu)^T]$$
$$= \tfrac{1}{K}E[\tfrac{1}{K}\textstyle\sum_{i,j}(\mathbf{x}_i-\mu)(\mathbf{x}_j-\mu)^T] = \tfrac{1}{K}E[\tfrac{1}{K}\sum(\mathbf{x}_i-\mu)(\mathbf{x}_i-\mu)^T]$$
$$= \tfrac{1}{K}\mathbf{\Sigma}_\mathbf{x}.$$

Johdetun tulosten avulla voidaan laskea neliöllisen virheen odotusarvo,

$$e^2 = \tfrac{1}{dim(\mathbf{x})}E[\|\mathbf{m}_K - \mu\|^2] = \tfrac{1}{dim(\mathbf{x})}E[tr((\mathbf{m}_K-\mu)(\mathbf{m}_K-\mu)^T)]$$
$$= \tfrac{1}{dim(\mathbf{x})}tr(\mathbf{\Sigma}_{\mathbf{m}_K}) = \tfrac{1}{dim(\mathbf{x})K}tr(\mathbf{\Sigma}_\mathbf{x}) \leq \tfrac{1}{K}\lambda_{max}.$$

Jäljellä olevaa neliöllistä virhettä voidaan siis arvioida laskemalla kovariassimatriisin diagonaaliset arvot tai pyrkimällä arvioimaan maksimiominaisarvoa. Näistä edellinen voidaan tehdä laskennan aikana ja jälkimmäistä ylärajaa voidaan pyrkiä arvioimaan analyyttisesti. Lasketun valuma-aluekartan $\mathbf{P}$ yksittäisten pisteiden varianssi on aiemman tuloksen mukaisesti $p(1-p)$. Koska valuma-alueeseen varmasti kuulumattomissa $p = 0$ ja kuuluvissa $p = 1$ alueissa, yksittäisten pisteiden varianssi on nolla, on koko kartan varianssi kytköksissä valuma-alueen piirin pituuteen L. Olettamalla valuma-alueen, jonka pinta-ala on A, olevan ympyrän muotoinen tulee piiriksi $L = 2\pi r = 2\sqrt{\pi A}$. Koko valuma-aluekartan varianssi on siis luokka $O(L)$, joka on aina suurempi kun valuma-aluekartan kovarianssimatriisin suurin ominaisarvo. Näin neliöllisen virheen kasvulle saadaan yläraja arvio

$$e^2 \leq O(\sqrt{A}/K).$$

Koska valuma-alueen pinta-alaa A:ta voidaan vielä arvioida ylhäältä päin koko korkeusmallin pinta-alaa käyttämällä $D^2 \geq A$, on tarpeellinen iteraatiokertojen määrä suhteessa suorakulmaisen korkeusmallin reunan pituuteen $K = O(D)$. Esimerkiksi 1000×1000 kokoisen alueelle valuma-aluekartan $\mathbf{P}_K$ tarkka laskeminen vaatii noin 10 kertaa enemmän laskentaa kuin 100×100 kokoinen alue.

3.3 Ulottuvuuksien määrän vähentäminen

Lineaarinen pääkomponenttianalyysi[Hay 98] on tapa kuvata muuttujat $\mathbf{x}$ (esimerkiksi valuma-alue $vec(\mathbf{X})$) lineaarisesti muuttujiksi $\mathbf{y} = \mathbf{A}\mathbf{x}$ siten, että kuvaus $\mathbf{A}$ diagonalisoi kovarianssimatriisin $\mathbf{\Sigma}_\mathbf{y}$. Tällöin muuttujat $\mathbf{y}$ ovat dekorreloitu eli $Cov(\mathbf{y}_i, \mathbf{y}_{j\neq i}) = 1$. Menetelmä perustuu symmetrisen kovarianssimatriisin $\mathbf{\Sigma}_\mathbf{x}$ ominaisarvokehitelmään$\mathbf{\Sigma}_\mathbf{x} = \mathbf{X}\mathbf{\Lambda}\mathbf{X}^T$. Kaavassa $\mathbf{X}$ on rotaatiomatriisi $\mathbf{X}^T\mathbf{X} = \mathbf{I}$ ja $\mathbf{\Lambda}$ on diagonaalinen matriisi. Valitsemalla $\mathbf{A} = \mathbf{X}^\mathbf{T}$ tulee $\mathbf{y}$:n kovarianssimatriisiksi

$$\begin{aligned}\mathbf{\Sigma_y} &= E[(\mathbf{y}-\mu_\mathbf{y})(\mathbf{y}-\mu_\mathbf{y})^T] = E[\mathbf{A}(\mathbf{x}-\mu_\mathbf{x})(\mathbf{x}-\mu_\mathbf{x})^T\mathbf{A}^T] \\ &= \mathbf{A\Sigma_x A}^T = \mathbf{X}^T(\mathbf{X\Lambda X}^T)\mathbf{X} = \mathbf{\Lambda}.\end{aligned}$$

Tätä tulosta voidaan ja myös usein käytetään $\mathbf{x}$:n ulottuvuuksien määrän vähentämiseen. Diagonaalinen $\mathbf{y}$:n korrelaatiomatriisi $\mathbf{\Lambda}$ sisältää diagonaalillaan muuttujien $\mathbf{y}$ varianssit. Asettamalla varianssit suuruusjärjestykseen $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_N$ ja asettamalla näistä pienimmät $i > M$ nolliksi, voidaan $\mathbf{y}$:stä tiputtaa pois nämä vähän satunnaisuutta sisältävät ulottuvuudet. Pääkomponenttianalyysin avulla voidaan siis etsiä matalaulotteisempi aliavaruus, jonka kovarianssimatriisin ominaisarvohajotelman diagonaalisella matriisilla $\mathbf{\Lambda}$ on lähes saman verran varianssi. Näin voidaan korkeaulotteista laskentaa usein helpottaa tekemällä laskut matalaulotteisemmalla aineistolla $\mathbf{y}$. Tätä lähestymistapaa voidaan soveltaa myös keskiarvon konvergenssin arvioimiseen.

Aiemmin laskettujen tulosten perusteella on selvää, että keskiarvon konvergenssi on riippuvainen satunnaismuuttujien variansseista, jotka haluttaisiin nyt pääkomponenttianalyysin avulla keskittää mahdollisimman pieneen määrään muuttujia. Valuma-aluekarttojen varianssi on keskittynyt valuma-alueen epävarmoille reunoille, joiden koko on huomattavasti koko valuma-aluetta pienempi. Koska lineaarisella kuvauksella $\mathbf{Ax}$ pystytään näitä pisteitä helposti "poimimaan" $\mathbf{y}$:n muuttujiin, pitäisi ulottuvuuksien määrän vähentäminen olla helppoa. Aliavaruus, joka sisältää esimerkiksi 90% koko valuma-alueen $\mathbf{x}$ varianssista, on ilmeisesti huomattavasti alkuperäistä matala-ulotteisempi.

Ulottuvuuksien vähentämistä voidaan konvergenssin laskemisessa perustella seuraavasti. Olkoon $\mathbf{A}$ aikaisempi lineaarinen kuvaus alkuperäisistä muuttujista $\mathbf{x}$ matala-ulotteisemmille muuttujille $\mathbf{y}$,

$$\mathbf{y} = \mathbf{Ax}.$$

Kuvaus $\mathbf{A}$ on valittu niin, että pisteiden $\mathbf{y}$ varianssi on mahdollisimman suuri. Lisäksi on oletettu satunnaismuuttujien olevan yksinkertaisuussyistä nollakeskiarvoisia, mikä ei rajoita seuraavien tulosten yleisyyttä. Koska $\mathbf{A}$ on rotaatiomatriisi, pätee käänteisesti $\mathbf{x} = \mathbf{A}^T\mathbf{y} + \epsilon$, missä ϵ on muunnoksessa hävitetystä varianssista johtuva satunnaisvirhe. Tällöin keskiarvoestimaatti voidaan kirjoittaa muodossa

$$\mathbf{m}_K = \mathbf{A}^T\left(\tfrac{1}{K}\textstyle\sum \mathbf{y}_i\right) + \left(\tfrac{1}{K}\textstyle\sum \epsilon_i\right),$$

$$\begin{aligned}&Var[\mathbf{m}_K] = \\ &Var[\mathbf{A}^T\left(\tfrac{1}{K}\textstyle\sum \mathbf{y}_i\right)] + Var[\left(\tfrac{1}{K}\textstyle\sum \epsilon_i\right)] + 2Cov[\mathbf{A}^T\left(\tfrac{1}{K}\textstyle\sum \mathbf{y}_i\right), \left(\tfrac{1}{K}\textstyle\sum \epsilon_i\right)].\end{aligned}$$

Mikäli nyt $\mathbf{y}$:n keskiarvo tiedetään olevan tarkasti tiedossa ja se ei korreloi merkittävästi $\epsilon : n$ keskiarvon kanssa, on $\mathbf{x}$:n keskiarvon varianssi peräisin ainoastaan pieni varianssisesta muuttujasta ϵ. Pääkomponentti-analyysissä tämä pitää paikkaansa, sillä pääkomponenttien väliset muuttujat eivät korreloi keskenään $Cov[\mathbf{A}^T\left(\tfrac{1}{K}\textstyle\sum \mathbf{y}_i\right), \left(\tfrac{1}{K}\textstyle\sum \epsilon_i\right)] = 0$. Tuloksen perusteella voidaan siis käyttää konvergenssinestimointia huomattavasti matalaulotteisimpiin muuttujiin mikäli ne sisältävät suurimman osan alkuperäisten muuttujien varianssista.

Pääkomponenttianalyysin laskeminen korkeaulotteisesta aineistosta, kuten valuma-aluekartoista, on kuitenkin aiemmin kuvatulla ominaisarvokehitelmään perustuvalla menetelmällä hidasta, koska se vaatii kovarianssimatriisin $\mathbf{\Sigma_x}$ laskemista. Ominaisarvosuotimella voidaan arvioida suurin datassa oleva ominaisarvo ja siihen liittyvä vektori käyttämättä muistia enempää kuin $O(D)$. Poistamalla ratkaistun ominaisarvon mukainen suuntaa koko käytettävistä olevasta datasta $\{\mathbf{x}_i\}$, voidaan menetelmästä tehdä iteratiivinen niin, että saadaan ratkaistua approksimatiivisesti K suurinta ominaisarvoa ja vektoria. Käyttämällä nyt lisäksi kovarianssimatriisin diagonaalielementtiin liittyvää tulosta

$$tr(\mathbf{\Sigma_x}) = tr(\mathbf{X\Lambda X}^T) = tr(\mathbf{\Lambda X}^T\mathbf{X}) = \sum_i \lambda_i.$$

Voidaan positiivisten ominaisarvojen eli varianssien summa laskea arvioimalla vain kovarianssimatriisin diagonaalisten elementtien summaa eli $\sum_i Var[x_i]$. Näin voidaan iteratiivisessa ominaisarvosuotimessa ratkaista varmasti p:n prosentin varianssialiavaruus.

Ominaisarvosuotimen 1. eigenfilterin tarkempi teoria on esitelty lähteessä [Hay 98]. Toteutuksessa suurimman ominaisarvon arvioimiseksi käytettiin valmista ominaisarvosuodinta, jota muokattiin arvioimaan p% varianssiavaruus ratkaisemalla datasta korkeavarianssisia vektorisuuntia kunnes p:n prosentin tavoite on saavutettu. Myöhemmin esiteltävien tulosten mukaan menetelmä toimi hyvin valuma-alueilla, joista on helppo poimia korkeavarianssiset muuttujat muutamaan $\mathbf{y}$:n muuttujaan. Näin voidaan seuraavaksi johdettava konvergenssinmonitorointi menetelmä saada toimimaan myös valuma-alue laskennassa. Menetelmä ei kuitenkaan toimi hyvin mikäli havainnot $\{\mathbf{x}_i\}$ sisältävät aidosti korkeaulotteista, riippumatonta satunnaisuutta eli varianssia, jota ei voida lineaarisesti kuvata matalaulotteisempaan aliavaruuteen. Käytetyn ominaisarvosuotimen lähdekoodi on annettu työn liitteissä.

3.4 Moniulotteinen konvergenssin arviointi

Kappaleessa 3.2 nähtiin konvergenssiarvioinnin vaativan korkeusmallien pisteiden korrelaatioiden eli kovarianssimatriisin $\Sigma_\mathbf{x}$ tai ainakin sen suurimman ominaisarvon λ_{max} arvioimista. Kappaleessa ei kuitenkaan johdettu tilastollisesti kunnollista menetelmää tuloksen oikeellisuuden arvioimiseksi, lisäksi edellisessä kappaleessa jätettiin huomioimatta eri iteraatiokertojen välisten keskiarvoestimaattien välinen korrelaatio, jolla voidaan näyttää olevan huomattava vaikutus laskettujen luottamusvälien suuruuteen [Rob 04].

Luvussa 2 johdettiin menetelmä yksiulotteisen keskiarvon luottamusvälin arvioimiseksi. Kappaleen 2 menetelmä voidaan laajentaa vektoriarvoisille muuttujille $\{\mathbf{x}_i\}$. Tällöin keskiarvoprosessin analyyttisen käsittelyn kannalta kokonaisjakaumana on helpointa käyttää matriisijakaumaa. Lisäksi hyvin korkeaulotteisten muuttujien käsittelemiseksi joudutaan luottamusvälin laskentaan yksinkertaistamaan käyttämällä approksimointeja, joilla luottamusväliä voidaan arvioida ulottuvuuksien määrän kasvaessa suureksi.

Luottamusvälin analyyttisen käsittelyn vaatimaksi matriisijakaumaksi soveltuu hyvin matriisinormaalijakauma [Gup 99]. Matriisinormaalijakauma on vektoriarvoisen normaalijakauman epätäydellinen yleistys matriiseille. Matriisi $\mathbf{X}(n \times p)$ on matriisinormaalijakautunut keskiarvolla $\mathbf{\Pi}$, jos sen jakauma on

$$\mathbf{X} \sim \frac{e^{-\frac{1}{2}tr\left[\mathbf{\Sigma}^{-1}(\mathbf{X}-\mathbf{\Pi})\mathbf{\Psi}^{-1}(\mathbf{X}-\mathbf{\Pi})^T\right]}}{(2\pi)^{d^2/2}|\mathbf{\Sigma}|^{d/2}|\mathbf{\Psi}|^{d/2}},$$

mikä voidaan palauttaa vektoriarvoikseksi jakaumaksi

$$vec(\mathbf{X}^T) \sim \mathcal{N}(vec(\mathbf{\Pi}^T), \mathbf{\Sigma} \otimes \mathbf{\Psi}).$$

Kaavassa merkintä $vec(\cdot)$ tarkoittaa matriisin vektorointia niin, että matriisin arvot asetetaan vektoriin sarakkeittain. Kroneckerin tulon $\otimes$ määritelmä $m \times n$ ja $p \times q$ kokoisille matriiseille $\mathbf{A}$ ja $\mathbf{B}$ on alla olevan mukainen $mp \times nq$ kokoinen matriisi.

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & .. & a_{1n}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & .. & a_{2n}\mathbf{B} \\ .. & .. & .. & .. \\ a_{m1}\mathbf{B} & a_{m2}\mathbf{B} & .. & a_{mn}\mathbf{B} \end{bmatrix}$$

Normaalimatriisin mukaisilla matriiseilla jokaisen osamatriisin ja elementin jakauma on myös normaalijakautunut ja monet vektorinormaalijakaumaa koskevat tulokset voidaan luonnollisella tavalla laajentaa matriisinormaalijakaumaa koskevaksi. Matriisin elementtien x_{ij} ja x_{kl} välinen kovarianssi on

$$Cov[x_{ij}, x_{kl}] = \mathbf{\Sigma}(i,k)\mathbf{\Psi}(j,l).$$

Tästä tuloksesta nähdään $\mathbf{\Sigma}$ - termin kertoimien kuvaavan rivien välistä korrelaatiota ja $\mathbf{\Psi}$ - termin kertoimien kuvaavan sarakkeiden välistä korrelaatiota. Tämä jakauman parametrisointi sopii täydellisesti vektoriarvoisen keskiarvonestimointiprosessin konvergenssin arviointiin.

Merkitään vektoriarvoista havaintosarjaa $\mathbf{x}_1, \mathbf{x}_2, .., \mathbf{x}_K$, jolloin keskiarvon $\mu_{\mathbf{x}}$ estimaattori on

$$\mathbf{m}_K = \frac{1}{K}\sum \mathbf{x}_i \ , \ i = 1...K.$$

Kuten yksiulotteisessa tapauksessa voidaan keskiarvojen kovarianssimatriisi laskea

$$\mathbf{S}_{max(K,L)} = E[(\mathbf{m}_K - \mu_{\mathbf{x}})(\mathbf{m}_L - \mu_{\mathbf{x}})^T] = \frac{1}{max(K,L)}\mathbf{\Sigma}_{\mathbf{x}}.$$

Tällöin voidaan keskiarvoprosessimatriisilla $\mathbf{M}_K = [\mathbf{m}_1\mathbf{m}_2...\mathbf{m}_K]$ nähdä olevan matriisinormaalijakauma $\mathcal{N}(\mathbf{\Pi}, \mathbf{\Sigma}, \mathbf{\Psi})$, jossa keskiarvona on

$$\mathbf{\Pi} = \begin{bmatrix} \mu & \mu & ... & \mu \end{bmatrix}$$

ja kovarianssimatriisit ovat $\mathbf{\Psi}(k,l) = \frac{1}{max(k,l)}$ ja $\mathbf{\Sigma} = \mathbf{\Sigma}_{\mathbf{x}}$.

Konvergenssi tarkasteluissa on tarpeen kääntää kumpikin jakauman matriisi. Matriiseista $\mathbf{\Psi}$ voidaan kääntää samoin kuin yksiulotteisessa konvergenssin tarkastelussa. Korrelaatiomatriisin $\mathbf{\Sigma}$ kääntäminen on korkeaulotteisissa tapauksissa sen sijaan laskennallisesti liian hidasta. Oikeaoppisesti toimittaessa Wishart-jakautunut matriisi jouduttaisiin arvioimaan mittauksista, eikä sen symmetrisyyttä lukuunottamatta tarvitse noudattaa mitään laskentaa helpottavia säännönmukaisuuksia.

Konvergenssiarvion laskeminen

Kun kumpikin jakauman matriisi on tiedossa pystytään konvergenssia arvioitaessa hyödyntämään vektorijakaumista matriisijakaumille yleistettävissä olevaa tulosta [Gup 99]:

$$tr\left[\mathbf{\Sigma}^{-1}(\mathbf{M}_K - \mathbf{\Pi})\mathbf{\Psi}^{-1}(\mathbf{M}_K - \mathbf{\Pi})^T\right] \sim \chi^2_{np},\ np = DK$$

$$\text{ja} \sim npF_{np,\nu} \text{ jos } \mathbf{\Sigma} \text{ matriisi ei ole tiedossa ja } \mathbf{\Sigma} \sim W_\nu(\mathbf{S}).$$

Kaavassa χ^2_{np} on kshiin neliön jakauma, $npF_{np,\nu}$ on np skaalattujen muuttujien Fisherin F-jakuama ja $W_\nu(\mathbf{S})$ on Wishart-jakauma [Gup 99], joka on odotusarvon $E[\mathbf{x}\mathbf{x}^T]$ jakauma kun kovariassinmatriisia on arvioitu $\nu = K$:lla näytteellä.

Kuten vektoriarvoisessa tapauksessa myös yllä kuvatussa jakaumatuloksessa on oletettu matriisin Σ estimaatin olevan riippumaton $\mathbf{M}$:stä, mikä voidaan taata käyttämällä erillisiä näytteitä matriisin ja keskiarvon arviointiin. Estimointiprosessin luottamusväliä voidaan siten arvioida samoin kuin yksiulotteisilla muuttujilla. Kun d_K on jakauman kertymäfunktiosta $F(x)$ ratkaistu $p\%$ raja-arvo $d_K = F^{-1}(p)$ neliölliselle virhetermille voidaan laskea ratkaisu:

$$tr\left[\mathbf{\Sigma}^{-1}(\mathbf{M}_K - \mathbf{\Pi})\mathbf{\Psi}^{-1}(\mathbf{M}_K - \mathbf{\Pi})^T\right] \leq d_K$$

$$tr\left[\mathbf{\Sigma}^{-1}\mathbf{\Pi}\mathbf{\Psi}^{-1}\mathbf{\Pi}^T - 2\mathbf{\Sigma}^{-1}\mathbf{M}_K\mathbf{\Psi}^{-1}\mathbf{\Pi}^T\right] + tr\left[\mathbf{\Sigma}^{-1}\mathbf{M}_K\mathbf{\Psi}^{-1}\mathbf{M}_K^T\right] \leq d_K.$$

Tässä voidaan käyttää tulosta $\mathbf{\Psi}^{-1}\mathbf{1} = \begin{bmatrix} 0 \\ 0 \\ \dots \\ K(K+1) \end{bmatrix}$, jolloin

$$\mathbf{\Psi}^{-1}\mathbf{\Pi}^T = \mathbf{\Psi}^{-1}\begin{bmatrix} \mu^T \\ \mu^T \\ \dots \\ \mu^T \end{bmatrix} = \begin{bmatrix} \mathbf{0}^T \\ \mathbf{0}^T \\ \dots \\ K(K+1)\mu^T \end{bmatrix}$$

ja laskua voidaan jatkaa

$$tr\left[K(K+1)\mathbf{\Sigma}^{-1}\mu\mu^T - 2K(K+1)\Sigma^{-1}\mathbf{m}_K\mu^T\right] \leq d_K - tr\left[\mathbf{\Sigma}^{-1}\mathbf{M}_K\mathbf{\Psi}^{-1}\mathbf{M}_K^T\right]$$

$$(\mu - \mathbf{m}_K)^T\mathbf{\Sigma}^{-1}(\mu - \mathbf{m}_K) \leq$$
$$\mathbf{m}_K^T\Sigma^{-1}\mathbf{m}_K + \frac{1}{K(K+1)}\left\{d_K - tr\left[\mathbf{\Sigma}^{-1}\mathbf{M}_K\mathbf{\Psi}^{-1}\mathbf{M}_K^T\right]\right\} = R^2$$

$$\|\mu - \mathbf{m}_K\|^2_\Sigma \leq R^2, \text{ missä}$$

$$R = \sqrt{\frac{d_K}{K(K+1)} + \mathbf{m}_K^T\Sigma^{-1}\mathbf{m}_K - \frac{1}{K(K+1)}tr\left[(\mathbf{\Sigma}^{-1}\mathbf{M}_K)(\mathbf{\Psi}^{-1}\mathbf{M}_K^T)\right]}.$$

Luottamusväli voidaan siten ilmaista yksinkertaisessa muodossa Mahalanobis-etäisyyden avulla. Mikäli myös korrelaatiomatriisin $\mathbf{\Sigma}$ suurin ominaisarvo λ_{max} on tiedossa, voidaan laskea yläraja luottamusvälin suuruudelle,

$$\|\mu - \mathbf{m}_K\|_2 \leq \sqrt{\lambda_{max}} R.$$

Kaava antaa maksimaalisen etäisyyden origosta D-ulotteisen ellipsin $\|\mu - \mathbf{m}_K\|_\Sigma^2 = R^2$ pinnalle. Euklidisena etäisyysmittana se antaa siten ylärajan yhteenlasketulle virheelle,

$$\|\mu - \mathbf{m}_K\|_2 = \sqrt{\sum_i (\mu_i - \mathbf{m}_{K,i})^2}.$$

Tuloksesta voidaan myös laskea keskiarvo neliölliselle virheelle epävarmalla valuma-alueen reunan alueella. Aiemman tuloksen mukaan valuma-alueen piirin voidaan arvioida olevan samaa suuruusluokkaa korkeusmallin pinta-alan neliöjuuren kanssa. Mikäli tämä oletus pitää likimain paikkaansa voidaan laskea laskennallinen keskivirhearvio

$$m_e = \frac{1}{\sqrt{dim(\mathbf{m}_K)}} \|\mu - \mathbf{m}_K\|_2 = \frac{1}{\sqrt{dim(\mathbf{m}_K)}} \sum_i (\mu_i - m_{K,i})^2.$$

Vaihtoehtoinen tapa mitata virhettä on laskea maksimaalinen virhe yhdessä ulottuvuudessa. Vaitsemalla $\mu_i = m_{K,i}$, kun $i \neq j$, saadaan kokonaisvirhe R kohdistettua vain yhteen muuttujaan ja konvergenssiarvion kaava supistuu muotoon

$$|\mu_j - m_{K,j}| \leq R\sqrt{\mathbf{\Sigma}(j,j)}.$$

Tämä valinta antaa siten suurimman vaihteluvälin yksittäiselle muuttujalle. Kun kovarianssimatriisista etsitään sen suurin diagonaalinen arvo σ_{max}^2, saadaan laskettua suurin jäljellä olevan virheen yläraja kaikille muuttujille.

Yhdistelemällä luvussa esitettyjä tuloksia saadaan konvergenssia arvioitua laskemalla korkeauloteisista valuma-alueista aluksi matalaulotteisempi 90% koko muuttujissa olevista variansseista sisältävä aliavaruus. Näin muuttujien määrä saaadaan valuma-aluelaskennassa tiputettua 10^6 ulottuvuudesta alle sataan. Tämä ei ole yllätys, koska korkeavarianssisiin aliavaruuksiin tarvitsee poimia pisteitä vain valuma-alueen epävarmoilta reunoilta. Laskentavaihe on hidas, mutta käyttämällä ominaisarvosuodinta pääkomponenttianalyysin laskemiseen on tarvittava muistin määrä D-ulotteisilla muuttujille vain $O(D)$. Tämän jälkeen voidaan konvergenssia arvioida neliöllisen virhekaavan $\|\mu - \mathbf{m}_K\|_2^2 \leq \lambda_{max}^2 R^2$ avulla. Suurimman ominaisarvon arvio saadaan ominaisarvosuotimen laskemasta arviosta. Matalaulotteisissa aliavaruudessa konvergenssin arviointi on nopeaa ja näytteistä laskettavat matriisit $\mathbf{M}_K$, $\mathbf{\Sigma}$ ovat pieniä.

Luku 4

Algoritmien hajauttaminen

Edellisessä kappaleessa esitettyjen arvioiden perusteella keskiarvotodennäköisyyksien, joilla pisteet kuuluvat valuma-alueeseen, laskeminen voi vaatia suuria määriä valuma-aluenäytteitä. Tämän lisäksi suuret korkeusmallit vaativat paljon muistia. Jotta simulointi saataisiin laskettua tarkasti myös suurilla korkeusmalleilla, voidaan simulointia nopeuttaa rinnakkais- ja hajautetun laskennan keinoin. Työssä keskityttiin algoritmien hajauttamiseen useamman koneen kesken, jolloin laskentakoneina l. solmuina voidaan käyttää myös normaaleja PC koneita ja suurimman käsiteltävissä olevan korkeusmallin koko voi olla yhden koneen muistimäärää suurempi.

Luvussa johdetaan aiemmin luvussa kaksi esitettyjen valuma-alueiden laskentaan ja Monte Carlo simulaatioon liittyvien algoritmien hajautetut toteutukset. Laskennan kannalta tärkeimmän ja hitaimman l. kuoppientäyttövaiheen osalta esitetään oma todistus Wangin nopealle kuoppientäyttöalgoritmille, josta on helppo todistaa kehitetyn hajautetun algoritmin toimivuus.

4.1 Yleistä

Hajautuksen lähtökohdaksi otettiin korkeusmallin jakaminen solmukoneiden muistimäärän mukaisiin paloihin. Jos solmukoneessa on muistia laskentaa varten käytettävissä n. 512 MB ja rinnakkaistetut laskentavaiheet vaativat suurimmillaan 20 korkeusmallin pitämistä yhtä aikaa muistissa, niin suurin käsiteltävissä oleva korkeusmallin koko ilman muistin loppumista on 2500 x 2500 pistettä.

Aiemman, laskentavaiheita ja algoritmejä kuvaavan kappaleen mukaisesti valuma-alueiden laskenta voidaan jakaa itsenäisiin osiin:

- virhepinnan luonti,
- kuoppien täyttö,
- varsinaisen valuma-alueen laskeminen korkeusmallista, sekä
- keskiarvon laskenta ja tuloksien konvergenssin arviointi.

Toteutuksessa pyritään hajauttamaan nämä algoritmit ja kehittämään heuristiikka korkeusmallin jakamiseksi solmukoneiden kesken.

4.2 Virhepinnan luonti

Virhepinnan laskennan hajauttamisen perustaksi otettiin aiemmin esitetty prosessikonvoluutio. Konvoluution hajauttaminen onnistuu helpohkosti, jos jompikumpi konvoloitavasta alueesta on huomattavasti lyhyempi toiseen nähden. Paikallisesti korreloituneessa virhepinnassa tämä ehto toteutuu. Aiemmin esitetyn mukaisesti konvoluutio voidaan toteuttaa nopealla Fourier-muunnoksella, joka vaatii $N \times M$ kokoisella pinnalla $O(MNlog(MN))$ operaatiota.

Hajautettussa konvoluutiossa virhepinnan luonti tapahtui seuraavasti. Aluksi kukin laskentasolmu generoi oman korreloimattoman virhepintansa. Tämän jälkeen kukin solmu vastaanottaa suotimen koon mukaan lasketun alueen naapurisolmujensa korreloimattomista virhepinnoista ja lähettää vastaavasti omat reuna-alueensa naapurisolmuille. Konvolointi voidaan tämän jälkeen tehdä normaalisti lisäämällä naapurisolmuista saadut virhepinnan palat alueen reunoille.

Konvoloimalla esimerkiksi 2048x2048 kokoista aluetta 129x129 (2x64 + 1) kokoisella suotimella, tarvitaan alueen reunoilla 64 pisteen levyinen alue naapurisolmujen virhepintaa. Tällöin alueen kooksi tulee 2112x2112 (2048 + 64) ja kaikkien kahdeksan naapurin ympäröimä laskentasolmu vastaanottaa korreloimatonta virhepintaa yhteensä 4*(64*64) + 4*2048*64 = 540672 pistettä. Yleisemmin $X \times Y$ kokoisella suotimella ja $W \times H$ kokoisella alueella tietoa siirretään yhdelle, koko laskenta-alueen keskellä olevalle laskentasolmulla ja -alueelle $4XY + 2XH + 2YW$ pistettä. Kun koko korkeusmallin alue on jaettu yhtäsuuriin $\frac{W}{K} \times \frac{H}{K}$ kokoisiin alueisiin, voidaan vastaavasti laskea arvio minimaaliselle tiedonsiirron määrälle

$$O(max(H, W)K).$$

Näin tehdyn jaon mukaan korkeusmallin alue on jaettu K^2:een osa-alueeseen. Tuloksen perusteella tiedonsiirron tarve kasvaa korkeusmallin pidemmän sivun ja tarvittavien laskentasolmujen määrän mukaan. Tuloksen todistus, samoin kuin kaava, jolla voidaan arvioida optimaalista K:n arvoa, löytyy liitteistä.

Taulukko 4.1: Hajautettu virhepinnan luonti

```
Funktio virhepinta = LaskeVirhepinta(Alue A, Suodin suodin)
  ;; suodin(2·suodin.X + 1, 2·suodin.Y + 1)
  ;; suodin.koko() = (2·suodin.X + 1)(2·suodin.Y + 1)
  Taulukko pinta(A.X+2*suodin.X, A.Y+2*suodin.Y)

  ;; korreloimaton virhepinta
  TeeNormaalijakaantunutPinta(
    pinta[suodin.X .. (suodin.X+A.X),
          suodin.Y .. (suodin.Y+A.Y)])

  Jokaisella naapurialueelle A_i
    LähetäAlue(pinta[(A_i + suodin.koko()) ∩ A], i)
    VastaanotaAlue(pinta[(A + suodin.koko()) ∩ A_i], i)
  LopetaJokaisella
```

```
    ;; konvolointi (suora FFT:n soveltaminen)
    S = FFT(LaajennaNollilla(suodin, suodin.koko()+pinta.koko()))
    P = FFT(LaajennaNollilla(pinta, pinta.koko()+suodin.koko()))
    Jokaiselle naapuripisteelle p
      SP[p] = S[p]P[p]
    LopetaJokaiselle

    pinta = FFT^-1(SP)

    ;; leikataan oikean kokoinen alue lasketusta alueesta
    virhepinta = pinta[suodin.X .. (suodin.X+A.X),
                       suodin.Y .. (suodin.Y+A.Y)]
LopetaFunktio
```

4.3 Kuoppien täyttäminen

Aikasemman luvussa kaksi annettujen tulosten mukaan on normaalissa, hajauttamattomassa laskennassa nopeinta käyttää Wangin/Liun kuoppientäyttöalgoritmiä, jolla kuoppien täyttö saadaan yksinkertaisella tavalla pysymään laskennallisesti tehokkaana. Menetelmä voidaan myös hajauttaa helposti toisin kuin monimutkaisempi Jensonin/Dominguen algoritmi. Wangin/Liun algoritmin laskennallinen kompleksisuus $N \times M$ kokoiselle korkeusmallille on

$$O(NM * log(NM)) \approx O(N^2 log(N)).$$

Luvussa 2.1.2. esitetyn algoritmin perusideana on lähteä etenemään korkeusmallin reunoilta sisäänpäin kohti keskustaa ja nostaa korkeusmatriisin pistealkioita, jotka eivät voi valua korkeusmallin reunalle eli ovat kuopassa. Menetelmä voidaan näyttää toimivan oikein mikäli reunapistejoukkoon lisätään aina reunan matalin piste. Algoritmin totetuksessa lähdetään liikkeelle korkeusmallin reunasta $\partial A = \partial A_0$ ja nopeuden kannalta tärkeä joukko ∂A toteutetaan prioriteettijonona, jolloin sekä lisäys- että poisto-operaatiot ovat mahdollisimman nopeita.

Seuraavaksi todistetaan Wangin/Liun algoritmin oikeellisuus alkuperäisestä lähteestä poikkeavalla tavalla. Tämän jälkeen käydään läpi mahdollisia hajauttamistapoja ja todistetaan valitun menetelmän toimivuus hajauttamattoman algoritmin todistusta laajentamalla.

4.3.1 Wangin ja Liun algoritmin todistus

Olkoon A ja B käsiteltyjen reunapisteiden ja käsittelemättömien sisäpisteiden joukot. $K = A \cup B$ on joukko korkeusmallin kaikille pisteille ja ∂A_0 on joukko korkeusmallin K reunapisteille, $h(\mathbf{x})$ ja $H(\mathbf{x})$ ovat alueen alkuperäinen ja korotettu korkeus pisteessä $\mathbf{x}$, laskennan aluksi $H(\mathbf{x}) = h(\mathbf{x})$. Funktio $d(\cdot, \cdot)$ on etäisyysmitta pisteiden välillä. Etäisyys $d(\mathbf{x}, \mathbf{y})$ on lyhin reitti hilan pisteiden välillä, kun etäisyys kaikkiin pisteen naapuripisteisiin, eli myös nurkkapisteisiin $(x+1, y+1)$, on yksi. Lisäksi joukon A rajapisteet ∂A määritellään seuraavasti

$$\partial A = \{\mathbf{x} \in A \mid \exists \mathbf{n} \in B\, d(\mathbf{x}, \mathbf{n}) = 1\}.$$

Laskennan aikana kaikille joukon A pisteille löytyy aina laskeva polku P_x korkeusmallin reunalle eli laskennan aikana pätee invariantti

$$\forall \mathbf{x} \in A \exists P_{\mathbf{x}} = \{\mathbf{x}_i\} \text{ s.e.}$$

$$\mathbf{x}_0 \in \partial A_0 \wedge \mathbf{x}_N = \mathbf{x} \wedge d(\mathbf{x}_{i-1}, \mathbf{x}_i) = 1 \wedge H(\mathbf{x}_{i-1}) \leq H(\mathbf{x}_i),$$

missä $P_{\mathbf{x}}$ on polku korkeusmallin reunalta pisteeseen $\mathbf{x}$ ja polun korkeus $h(P_{\mathbf{x}}) = max\{H(x_i\}$ on korkein reitillä oleva piste. Tämä invariantti on voimassa laskennan aluksi, kun $A = \partial A_0$.

Laskeville, pisteeseen $\mathbf{x}$ päätyville poluille $P_{\mathbf{x}}$ koko polun korkein piste on polun loppupiste eli $h(P_{\mathbf{x}}) = H(\mathbf{x})$. Lisäämällä nyt joukkoon A koko rajapistejoukon ∂A matalimman pisteen $\mathbf{z}$ ympäristö $N(\mathbf{z})$ ja nostamalla nämä pisteet tarvittaessa ∂A:ssa ja A:ssa olevan isäpisteen $\mathbf{z}$ tasolla, on myös näillä naapuripisteillä $N(\mathbf{z}) = \{\mathbf{y} \mid d(\mathbf{y}, \mathbf{z}) = 1\}$ laskeva polku alueen reunalle

$$\mathbf{y} \in N(\mathbf{z}),\ P_{\mathbf{y}} = P_{\mathbf{z}} \cup \{\mathbf{y}\},$$

$$H(\mathbf{y}) = max(h(\mathbf{y}), h(P_{\mathbf{z}})) = max(h(\mathbf{y}), H(\mathbf{z})).$$

Kun isäpisteenä on koko reunan matalin piste, on myös tehty korotus minimaalinen. Jos näet löytyisi toinen yhtä korkea polku $h(P_{\mathbf{w}})$ joukosta B reunalle ∂A_0 tulee myös tämän polun kulkea reunan ∂A kautta, jolloin polun korkeus on aina $h(P_{\mathbf{w}}) \geq min(\partial A)$ ja pisteen $\mathbf{w}$ lisääminen alueeseen A saa aina aikaan vähintään yhtä suuren korotuksen pisteissä $P_{\mathbf{w}} \cap B$.

Selvästi näin tehdyt peräkkäiset korkeusmallin korotukset ja alueen A laajennukset laajentavat A:n koko alueen K kokoiseksi, jolloin $B = K \setminus A = \emptyset$. Lisäksi kaikki reitit reunoilta ∂A_0 alueen sisäpisteisiin ovat tehtyjen korotusten minimaalisuuden takia mahdollisimman matalia. Jos näet minimaalisesti lisätty piste $\mathbf{x}$ olisi lisätty myöhemmin laskennan aikana korkeusmalliin, olisi reunan matalin piste $min(\partial A)$ ja siten myös $h(P_{\mathbf{x}})$ mahdollisesti korkeampi ja koska toisaalta pisteen korotus on minimaalinen, ei se aiheuta minkään myöhemmin nostettavan pisteen $\mathbf{y}$ polun korkeuden $h(P_{\mathbf{y}})$ suurenemista suuremmaksi, kuin mitä olisi saavutettavissa jollakin muulla korkeusmallin pisteiden korotusjärjestyksellä.

4.3.2 Hajauttaminen

Wangin algoritmi on yksinkertainen ymmärtää, mutta sen hajauttaminen on ongelmallista. Koska laskettava korkeusmalli on jaettu eri laskentasolmujen välille, on järkevintä pyrkiä keskittämään tiettyyn alueeseen liittyvä laskenta yhteen solmuun. Tämä on kuitenkin hankalaa, koska nopeuden kannalta tärkeä joukon ∂A prioriteettijonotietorakenteen tulee sisältää tiedot koko laskenta-alueen reunapisteistä. Prioriteettijono jouduttaisiin pahimmillaan toteuttamaan hajautetusti. Yhteisen prioriteettijonon käyttäminen pakottaisi laskentasolmut lähettämään keskenään $O(log(\#CPU))$ viestiä [Man 05] jokaisen jonon lisäys- ja poisto-operaation yhteydessä. Prioriteettijonon synkronointi jouduttaisiin lisäksi tekemään jokaisen pisteen lisäyksen ja poiston yhteydessä, joten algoritmin suorittaminen vaatisi

$$O(log(\#CPU)MN) \approx O(log(\#CPU)N^2)$$

viestin lähettämistä laskentasolmujen välillä.

Vaihtoehtoinen tapa on pyrkiä hajauttamaan Dominguen algoritmiä, jossa kuoppien täytössä käytettäviä tietorakenteita muokataan etupäässä paikallisesti. Valuma-alueita yhdistettäessä on laskentasolmujen tällöin tarpeen lukita käyttöönsä vain välittömästi valuma-alueeseen kosketuksissa olevat paikalliset valuma-alueet. Koska valuma-alueet sijaitsevat etupäässä ko. laskentasolmun alueella, tietorakenteiden lukitseminen päivitystä varten voidaan tehdä suurimmalta osin paikallisesti itse laskentasolmussa. Tältä osin Dominguen algoritmi vaikuttaa sopivan hajautettavaksi Wangin algoritmia paremmin. Dominguen algoritmin muut vaiheet, erityisesti valumien kertymien laskenta, vaativat kuitenkin koko korkeusmallin läpikäynnin ja huomattavan määrän viestintää laskentasolmujen välillä. Tämän takia työssä päädyttiin käyttämään Wangin algoritmin muokattua versiota. Kehitetty hajautettu algoritmi laskee jotkin korkeusmallin pisteet moneen kertaan, mutta mahdollistaa paikallisten, vain laskentasolmujen osa-alueet kattavien prioriteettijonojen käytön.

Hajautettu kuoppientäyttöalgoritmi

Kuoppientäyttölaskennan hajautus toteutetaan siten, että jokaisella laskentasolmulla $i = 1..N$ on oma paikallinen alueensa A_i, $K = \bigcup A_i$, sekä oma reunapistejoukkonsa ∂A_i, $\partial A = \bigcup \partial A_i$ ja $\partial A_{i,0} = A_i \cap \partial A_0$. Algoritmi etenee hajautetusti joukon ∂A_i pisteistä eteenpäin spekulatiivisesti niin, että myöhemmin toisesta laskentasolmusta tuleva matalampi valumareitti ja siihen liittyvät mahdolliset pisteiden korkeuden muutokset, voivat aiheuttaa jo kerran käsiteltyjen pisteiden liian suurten korotusten $H(\cdot)$ mataloitumista. Hajautetussa laskennassa seurataan myös ei-miniminostopolkuja, joiden vaikutukset kuitenkin laskennan aikana korjautuvat.

Seuraavaksi esiteltävässä valuma-alue algoritmin todistuksessa ajatellaan joukoissa $\{\partial A_i\}$ olevien pisteiden olevan jonkun pisteen $\mathbf{x}$, mahdollisesti muun kuin miniminostovalumapolun tämän hetkinen loppupiste, josta voidaan edetä eteenpäin. Tärkeä osa-joukko näistä korkeusmallin reunalta ∂A_0 lähtevistä valumapoluista ovat minimaalisen korkeusmallin noston $h(P_\mathbf{x})$ tekevät valumapolut $P_\mathbf{x}$. Koska miniminostovalumapolut nostavat korkeusmallia mahdollisimman vähän, on algoritmi päätynyt oikeaan lopputulokseen kun jokaisen pisteen yksi minimivalumapolku $P_\mathbf{x}$ on käyty kokonaan läpi.

Aluksi vain korkeusmallin reunoilla $A_i \cap \partial A_0 \neq \emptyset$ sijaitsevilla laskentasolmuilla on rajapisteitä, joista laskenta lähtee liikkeelle. Tämän jälkeen jokainen laskentasolmu käy läpi rajapistejoukkoaan kuten hajauttamattomassa algoritmissä. Tämä tapahtuu kuitenkin sillä poikkeuksella, että mahdolliset pisteiden korkeuden muutokset ja pisteiden lisäykset rajapistejoukkoon ∂A_i lähetetään alueen A_i reunassa operoitaessa niitä vastaaviin naapurisolmuihin. Vastaavasti muista laskentasolmuista vastaanotetut pisteet käsitellään normaalisti ja lisätään rajajoukkoon ∂A_i, vaikka ko. piste olisikin jo käsitelty. Tämä kuitenkin sillä poikkeuksella, että jo kerran käsiteltyjä pisteitä, joita ei niihin etenemisen seurauksena laskettu, ei käsitellä uudestaan, eli ei lisätä rajajoukkoon ∂A_i.

Todistetaan edellä kuvatun algoritmin toiminta. Kullakin laskentasolmulla i on oma rajajoukkonsa ∂A_i, joka hajautetussa algoritmissä sisältää käsitelyssä olevien *mahdollisten* matalimpien reittien $P_\mathbf{z}$ tämän hetkisen kärkipisteen.

Joukko ∂A_i sisältää pisteitä, joihin korkeusmallin reunalta löytyy ainakin ko. pisteen korkeuden mukainen reitti. Pistejoukoissa ∂A_i pätee siten laskennan ajan invariantti

$$I_1\text{: } \forall \mathbf{x} \in \partial A_i : H(\mathbf{x}) \geq h(P_\mathbf{x}),$$

missä $h(P_\mathbf{x})$ on lopullisen, oikean valumapolun matalin piste. Uudelleenlaskentaa vaativia korkeuden muutoksia l. korkeuden $H(\mathbf{x})$ laskua tapahtuu kun toinen, matalampi valumareitti päätyy pisteeseen $\mathbf{x}$. Modifioidussa hajautetussa algoritmissä edetään paikallisesti joukon ∂A_i matalimman pisteen $\mathbf{z} \in \partial A_i$ naapuripisteisiin $\mathbf{y} \in N(\mathbf{z})$. Pisteisiin, joissa on jo vierailtu ainakin kerran, ja joihin eteneminen ei aiheuttaisi korkeuden laskemista, ei kuitenkaan edetä. Pisteet $\mathbf{y} \in N(\mathbf{z})$ lisätään pisteiden reunajoukkoihin $\{\partial A_i\}$, jonka seurauksena edettävien pisteiden tämän hetkiset valumapolut kulkevat pisteen $\mathbf{z}$ kautta eli

$$P_\mathbf{y} = P_\mathbf{z} \cup \{\mathbf{y}\}$$

ja pisteiden hetkellinen korkeus lasketaan seuraavasti:

$$H(\mathbf{y}) = max(h(\mathbf{y}),\ H(\mathbf{z})) \geq h(P_\mathbf{y}),$$

mikä voi olla mahdollisesti korkeampi kuin pisteen lopullisen valumapolun maksimikorkeus. Koska joukossa ∂A_i oleville pisteille on voimassa invariantti I_1, on se myös voimassa uusille pisteille $\mathbf{y} \in N(\mathbf{z})$.

Algoritmin oikeellisuus konvergenssissa, eli kun $\forall \partial A_i$: $|\partial A_i| = 0$, voidaan nyt näyttää todeksi. Aluksi solmuilla on omat reuna-aluepisteensä $\partial A_{i,0} = A_i \cap \partial A_0$. Reunapistejoukosta ∂A_0 alkavista poluista löytyy kaikille pisteille $\{\mathbf{x}\}$ yksi tai useampi pisteen miniminostopolku korkeusmallin reunalle. Mahdolliset miniminostopolut kuuluvat joukkoon

$$\mathbf{P_x} = \{P_\mathbf{xw} \mid \nexists P_\mathbf{xv}\ s.e.\ h(P_\mathbf{xv}) < h(P_\mathbf{xw}),\ \mathbf{v}, \mathbf{w} \in \partial A_0\},$$

missä $P_\mathbf{xw}$ on polku (joukko) pisteitä pisteestä $\mathbf{x}$ pisteeseen $\mathbf{w}$.

Todistetaan nyt, että ei voi päteä $\forall \partial A_i$: $|\partial A_i| = 0$, mikäli jonkin pisteen minimireittiä ei ole käyty läpi. Merkitään merkinnällä $P_\mathbf{x}(n)$, $n \geq 1$ korkeusmallin reunalta alkavan polun n:ää ensimmäistä alkioita. Koska selvästi $\forall P_\mathbf{x} \forall n$: $P_\mathbf{x}(n) \cap \partial A_0 \neq \emptyset$, on seuraava invariantti voimassa kaikilla polun pituuksilla n

$$I_2\text{: } \forall \mathbf{x} \exists P_\mathbf{x} \subset \mathbf{P_x} \exists \partial A_i : P_\mathbf{x}(n) \cap \partial A_i \neq \emptyset \vee H(\mathbf{x}) = h(P_\mathbf{x}).$$

Tämä invariantti on selvästi voimassa laskennan alussa ja jokaisen korkeusmallin päivityksen jälkeenkin. Kun edetään seuraavaan pisteeseen $\mathbf{x}_{n+1}$ polulla $P_\mathbf{w}(n)$, pätee ainakin jollekin naapuripisteelle $\mathbf{y}$ $H(\mathbf{y}) \geq H(\mathbf{x})$, koska kyseessä on miniminostopolku. Tällöin lisätään joko yksi polun piste johonkin joukkoon ∂A_i, törmätään toiseen minimipolkuun $P_\mathbf{w}$ tai ollaan polun lopussa, jolloin invariantin I_1 mukaisesti $H(\mathbf{x}) = h(P_\mathbf{x})$. Toiseen minimipolkuun törmääminen kuitenkin tarkoittaa, että invariantti on ko. korkeusmallin pisteen osalta voimassa jo toisen pidemmälle edenneen minimipolun ansioista. Näin ollen invariantti I_2 on aina voimassa, jolloin konvergenssista seuraa

$$\forall \partial A_i\colon |\partial A_i| = 0 \wedge I_2 \Rightarrow \forall \mathbf{x} : H(\mathbf{x}) = h(P_{\mathbf{x}}).$$

Konvergenssi voidaan toisaalta nähdä seuraavasti. Äärellisen kokoisella korkeusmallilla jokaisella polulla, joka ei käy toistamiseen samassa pisteessä, on rajallinen pituus. Näytetään nyt, että kaikki ei-nousevat valumapolut eivät vieraile kahdesti samassa pisteessä. Algoritmi ei etene jo vierailemaansa pisteeseen, koska $P_{\mathbf{x}}$:n jokaisen osapolun $P_{\mathbf{x}}(1..n) = P_{\mathbf{z}}$ päätepisteelle $\mathbf{z}$ pätee jossakin laskennan vaiheessa

$$\forall \mathbf{w} \in P_{\mathbf{x}}(1..n), \mathbf{w} \neq \mathbf{z}\colon H(\mathbf{z}) \geq H(\mathbf{w}).$$

Toiset pisteessä $\mathbf{w}$ vierailevat polut voivat vain laskea polun pisteitä, joten algoritmi ei etene koskaan jo vierailuksi merkattuun pisteeseen. Tämä pätee kuitenkin vain jos polun pisteen avaamishetkistä päätekorkeutta $H(\mathbf{x})$ ei ole myöhemmin laskettu jonkun toisen polun seurauksena. Tämä voidaan taata tallentamalla polun $H(\mathbf{x})$ korkeus polun laajennushetkellä mukaan reunajoukkoon $\partial A_i := \{(\mathbf{x}, H(\mathbf{x}))\}$ja käyttämällä tallennetua arvoa polulla edetessä. Näin jokaisen polun pituus on rajallinen ja algoritmi konvergoituu tilanteeseen $\forall \partial A_i$: $|\partial A_i| = 0$.

Taulukko 4.2: Hajautettu kuoppientäyttöalgoritmi

```
Funktio Korkeusmalli = TäytäKuopat(Korkeusmalli A_i)
  Taulukko Vierailtu, PrioriteettiJono UudetPisteet

  Jokaiselle korkeusmallin alueella A_i olevalle reunapisteelle b
    Korotus[b] = Korkeus[b]
    UudetPisteet.lisää(Korotus[b], b)
    Vierailtu[b] = Epätosi
  LopetaJokaiselle

  Toista jos |UudetPisteet| > 0
    (p, korkeus) = UudetPisteet.poistaMatalin()
    KäsitellytPisteet.lisää(p)
    ;; tarkistetaan onko pisteeseen jo matalampi reitti
    Jos Vierailtu[p] ∧ Korotus[p] < korkeus
      Seuraava ;; pistettä ei tarvitse käsitellä

    Jokaiselle p:n naapuripisteelle n
      Jos piste n ∉ A_i
        Lähetä (n, Korkeus[n]) naapurialueeseen
      Muutoin
        k = Maksimi(Korkeus[n], korkeus)

        Jos Vierailtu[n] = Epätosi
          Vierailtu[n] = Tosi
          Korotus[n] = k
          UudetPisteet.lisää((n,k))
        Muutoin Jos Korotus[n] > k
          Korotus[n] = k
          UudetPisteet.lisää((n,k))
        LopetaJos
      LopetaJos
    LopetaJokaiselle

    Vastaanota pisteitä (p, korkeus) naapurisolmuista
      k = Maksimi(Korkeus[p], korkeus)
      Jos Vierailtu[p] ∧ Korotus[p] < korkeus
        Seuraava ;; pistettä ei tarvitse käsitellä
      Korotus[p] = k
      UudetPisteet.lisää((n,k))
    LopetaVastaanota

  LopetaToista
LopetaFunktio
```

Algoritmin tehokkuus

Algoritmin laskennallinen tehokkuus riippuu suoraan ylimääräisistä pisteissä vierailukerroista ja hajautetussa laskennassa ennen muuta laskentasolmujen välillä lähetettyjen pisteiden määristä. Laskennallinen alaraja algoritmin nopeudelle on korkeusmallin osa-alueiden toisia osa-alueita koskettavien pisteiden määrä, joka korkeusmallin sisäalueille on osa-alueen sisäpiirin pituinen. Alueen sisäpiiriin kuuluvat ne pisteet, jotka kuuluvat alueeseen ja ovat sen reunalla. Sisäpiirin pituus suorakulmallisille alueilla on siten:

$$l(A) = 2 * [height(A) + width(A) - 2].$$

Minimaalinen tiedonsiirtojen määrä jaolle $K = \bigcup A_i$ on rajapinnan l pituus $l = \sum l(A_i) - l(K)$. Tiedonsiirtojen määrälle voidaan myös laskea yläraja, jokainen korkeusmallin pisteen valumapolku kulkee korkeintaan kerran jokaisen pisteen kautta. Näin ollen tiedonsiirtojen T ylärajaksi tulee NMl.

Rajoista $l \leq T \leq NMl$ nähdään että alueiden A_i valitsemisessa olisi hyvä minimoida alueiden väliset rajat. Koska ympyrä tunnetusti minimoi piirin ja pinta-alan suhteen, olisi se hyvä valinta, mikäli alue voitaisiin peittää ympyröillä ilman suurempia päällekkäisyyksiä alueitten välillä. Tätä on kuitenkin mahdotonta tehdä täydellisesti ja sen tekeminen mahdollisimman pienillä päällekkäisyyksilläkin olisi hankalaa. Tämän takia sopivampi jaottelu alueisiin ovat suorakaiteet, joista neliö minimoi piirin ja katetun pinta-alan suhteen.

4.4 Valuma-alueiden laskenta

Kappaleessa kaksi esitellyn valuma-alueiden laskenta-algoritmin hajauttaminen voidaan tehdä samalla tavoin kuin kuoppien täyttäminen. Korkeusmalli jaetaan laskentasolmujen kesken osiin $K = \bigcup A_i$, minkä jälkeen kukin laskentasolmu etenee normaalisti eteenpäin oman alueen pisteissään. Mikäli algoritmi päätyy oman alueensa reunaan, se siirtää ko. valumapisteen tiedot naapurialueelle, jonka alueesen ko. piste kuuluu. Hajautuksen pseudokooditoteutus on näkyvissä ohessa. Algoritmin skaalautuvuus ei varsinaisesti huonone hajautuksen takia. Tiedonsiirron määrään laskentasolmujen välillä pätevät samat yleisluontoiset tulokset kuin kuoppientäyttöalgoritmissä.

Taulukko 4.3: Hajautettu valuma-alueiden laskenta-algoritmi

```
Funktio ValumaAlue = LaskeValumaAlue(Korkeusmalli, A_i, Pisteet)
  Joukko B = Pisteet
  ValumaAlue = Pisteet

  Toista jos | B | > 0
    Piste p = B.poistaPiste()

    Jokaiselle naapuripisteelle n
      Jos n ∉ A_i
        Lähetä piste n laskentasolmulle j: n ∈ A_j
      Muutoin Jos Korkeusmalli[p] ≤ Korkeusmalli[n] ∧
```

```
                n ∉ ValumaAlue
              B.lisääPiste(n)
              ValumaAlue.lisääPiste(n)
            LopetaJos
          LopetaJokaiselle

          Vastaanota pisteitä p
            Jos p ∉ ValumaAlue
              B.lisääPiste(p)
            LopetaJos
          LopetaVastaanota

        LopetaToista

      LopetaFunktio
```

4.5 Korkeusmallien paloittelu ja tiedonsiirto

Kuten aiemmin on tullut jo ilmi, hyvä korkeusmallin pilkkomisalgoritmi kokoaa ensin laskentasolmuista pienempiä miniklustereita, joilla on tarpeeksi muistia toimia itsenäisenä kokonaisuutena eli riittävästi muistia koko korkeusmallin alueelle. Tämän jälkeen tulee korkeusmatriisi jaetaan jokaisessa miniklusterissa laskentasolmujen muistimäärän kokoisiin alueisiin niin, että alueiden välinen rajapinta minimoituu.

Koska ratkaisujen hyvyysjärjestys ei muutu vaikka rajapinnat laskettaisiin kahdesti tai tulokseen lisättäisiin koko korkeusmallin piiri $l(K)$, voidaan minimoida myös alueitten piirien summaa $L = \sum l(A_i)$. Muiden laskentavaihden yksinkertaistamiseksi on myös lähes välttämätön rajoittaa laskenta-alueet suorakaiteen kokoisiksi alueiksi.

Kyseessä on optimointiongelma, joka voidaan esittää abstraktimmin myös matemaattisessa muodossa. Olkoon $\mathbf{w} = \begin{bmatrix} w_1 & w_2 & .. & w_N \end{bmatrix}$ vektori painoarvoille, jotka kertovat kuinka suuren osuuden kokonaismuistin tarpeesta kukin laskentasolmu kattaa $0 \leq w_i \leq 1$, $\sum w_i = 1$. Olkoon lisäksi K joukko kokonaislukuarvoisia koordinaattipareja, jotka kattavat $N \times M$ kokoisen yhtenäisen alueen. Ongelmana on etsiä suorakulmaisista alueista $A = \{A_i\}$, koostuva alueen K peitto $K = \bigcup A_i$, s.e. $\mid A \mid = N$, $\forall i \neq j : A_i \cap A_j = \emptyset$, $w_i = \frac{|A_i|}{|A|}$ ja ratkaisu minimoi alueiden kokonaispiirin $L(A) = \sum l(A_i)$.

Vaikka ongelma vaikuttaa yksinkertaiselta, siihen ei työssä löydetty eikä keksitty yksinkertaista optimaalista ratkaisua, tämän takia päädyttiin heuristiseen menetelmään korkeusmallin paloittelemiseksi. Menetelmässä korkeusmalli pyritään pitämään mahdollisimman lähellä neliötä, joka minimoi suorakaiteen piirin suhteessa sen alaan.

Aluksi liitetään kaikki painot l. laskentasolmut koko korkeusmalliin $B_0 = K$. Tämän jälkeen korkeusmalleissa, joissa on enemmän kuin yksi paino $\{w_i\}$, jaetaan painot kahteen, mahdollisimman yhtäsuureen joukkoon, jossa pyritään minimoimaan joukkojen välinen painoero $\Delta = \mid \sum w_i \mid - \mid \sum w_j \mid$. Kun joukot on luotu, jaetaan varsinainen alue sen pidemmältä sivulta niin, että osa-alueiden

suuruus on suhteessa joukkojen painoihin. Tämän jälkeen painot ja solmut siirretään näin luotuihin osa-alueisiin. Tätä jatketaan kunnes jokaisessa alueessa on vain yksi solmu ja paino.

Myös tässä heuristisessa algoritmissä on ongelmansa. W:n painon jakaminen kahteen, mahdollisimman yhtä painavaan ryhmään vaatisi kaikkien 2^W vaihtoehdon läpikäymisen, joka kuitenkin on suurilla solmumäärillä liian hidasta. Siksi myös tämä osa-ongelma ratkaistiin heuristisesti. Aluksi painot järjestetään suuruutensa mukaiseen järjestykseen, jonka jälkeen ongelma ratkaistaan optimaalisesti N:n painon erissä ($2^N = pieni$, toteutuksessa käytettiin $N = 16$) suurimmista painoista pienimpiin siirtyen. Lopuksi todennäköisesti ei-optimaalista ratkaisua pyritään vielä parantamaan etsimällä yhden muuttujan siirtoja tai vaihtoja painojoukkojen välillä (1-0, 0-1, 1-1 vaihdot). Näiden parantelujen määrä rajoitettiin kokeilemalla sataan, jolla rajoitettiin laskenta-ajan kasvaminen. Kokeellisesti havaittiin näin saatujen ratkaisujen olevan hyviä, mutta huomattaviakin parannuksia voitaisiin joissakin tapauksissa saada analysoimalla suurempien muuttuja joukkojen vaihtoja painojoukkojen välillä (2-1, 1-2, 2-2, .. N-N vaihdot).

4.6 Tulosten keruu ja konvergenssin arviointi

Kun valuma-alueet on saatu laskettua hajautetusti, tulee tulokset vielä kerätä keskiarvon laskentaa varten. Myös tämä vaihe voitaisiin hajauttaa laskemalla keskiarvo eri alueille eri laskentasolmuissa ja yhdistämällä tulokset vasta laskennan lopuksi yhdeksi korkeusmatriisiksi $\mathbf{K}$. Tällä tavoin voitaisiin säästää jokaisella iteraatiokerralla tulosten lähettäminen yhdelle levypalvelimelle, joka verkon rakenteesta riippuen ei välttämättä pysty vastaanottamaan tietoa kuin yhdeltä laskentasolmulta kerrallaan. Vaikka ratkaisu tarjoaisi selviä etuja, toteutettiin tulosten keruu jokaisella iteraatiokerralla. Ratkaisuun päädyttiin, koska:

1. haluttiin varmistaa, että yhden laskentakoneen kaatuminen ei pilaa jo laskettuja tuloksia,
2. voidaan nopeasti havaita laskentakoneissa olevat ongelmat, eikä vasta laskennan lopuksi keskiarvoja kerättäessä,
3. estimaatin konvergenssin arviointi haluttiin yksinkertaisuus syistä pyrkiä tekemään keskitetysti koko korkeusmallin alueelle, vaikka se rajoittaakin laskennan skaalautuvuutta. Vaihtoehtoisena tapana olisi voitu pyrkiä arvioimaan tulosten tarkkuutta pienemmistä alueista lasketuista konvergenssituloksista.

Toteutetussa ohjelmassa tulosten keruuvaiheessa kerättiin laskennan tulokset kaikilta laskentasolmuilta keskuskoneelle, joka tallensi tulokset levylle. Sama laskentasolmu arvioi myös lasketun todennäköisyys keskiarvon konvergenssia.

Luku 5

Ohjelmistoarkkitehtuuri

Aiemmissa kappaleissa esiteltyjen ja johdettujen menetelmien ja tulosten pohjalta kehitettiin C++ ohjelma valuma-alueiden laskemiseksi. Kappaleessa kuvataan korkean tason ohjelmistoarkkitehtuuriset ratkaisut ja ohjelman tekninen toteutus.

5.1 Ohjelman rakenne

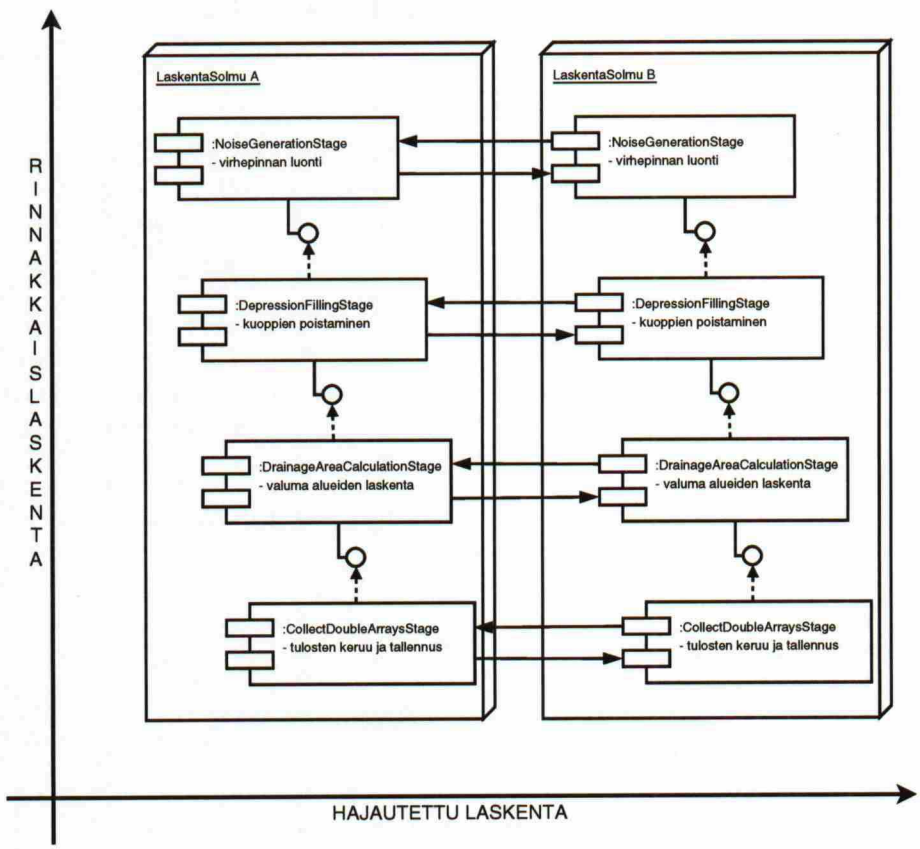
Aiemman kuvauksen mukainen hajautettu valuma-alueiden laskentaohjelma toteutettiin olio-orientoituneesti C++ ohjelmana Linux/Unix ympäristöön. Tällöin ohjelman aiemmin mainitut laskentaosat eli

- kerran alussa tapahtuva laskentasolmujen laskentakapasiteetin mukainen laskennan konfigurointi,
- virhepinnan luonti,
- kuoppien täyttö,
- varsinaisen valuma-alueen laskeminen korkeusmallista ja
- laskentatulosten kokoaminen

jaettiin olio-orientoituneesti itsenäisiksi kokonaisuuksiksi.

Pelkkien abstraktoinnin ja olio-orientoituneen luokkajaon lisäksi erillisistä laskentavaiheista muodostettiin liukuhihna- (pipeline) tyyppinen rakenne, missä eri vaiheiden laskenta tapahtuu erillisissä säikeissä. Nämä rinnakkaistetut laskentavaiheet voivat lisäksi toimia hajautetusti yhdessä toisten laskentasolmujen saman laskentavaiheen tai -osan kanssa. Näin ohjelman toiminta pyrittiin saamaan mahdollisimman tehokkaaksi.

Tunnetusti säikeiden ja prosessien käytöllä voidaan helposti hyödyntää laskentakapasiteettia paremmin myös silloin, kun jokin laskennan osavaihe jää odottamaan vastausta joltain I/O-laitteelta, kuten verkolta. Lisäksi liukuhihnoituksella saatiin myös ohjelman rakenteesta, sekä modulaarisempi, että moniprosessorisia laskentasolmuja paremmin hyödyntävä. Modulaarisuuden takia ohjelman toteuttaminen, testaaminen ja laajentaminen tulivat helpommaksi ja laskentaa voitiin helposti muuttaa lisäämällä tai poistamalla laskentavaiheita.



Kuva 5.1: Ohjelman rinnakkaisuus ja hajautus

Ohjelman hajatetun laskennan ja rinnakkaislaskennan suhdetta, sekä liukuhihnan toimintaa on havainnollistettu kuvissa 5.1 ja 5.2. Komentoriviltä käytettävän ohjelman komentoriviparametrit ovat taulukossa 5.1.

Taulukko 5.1: Ohjelman komentoriviparametrit

```
Commandline parameters:
mpidemanalysis <time> <inputhost> <dem_bin> <dpoints_noise> <outputhost> <results_file> [convcheck]
<time>          computation time in minutes
<inputhost>     DNS name for host where input data resides (use '_' for random node)
<dem_bin>       DEM file in BIN format (also .HDR file must exist)
<dpoints_noise> noise and drainage points (ASCII, 'NOISE <TYPE> <VAR> <PHI>' 'DP <X> <Y>' lines).
<outputhost>    DNS name for host to which results data is stored (use '_' for random node)
<results_file>  file to store results (RAW doubles), use results2xxx for conversion
[convcheck]     optional "-c[error]" can be used to give 95% confidence bound.
                Computation is stopped when approximate 95% confidence bound
                reaches the given error bound. If error bound is omited the
                95% error bound is reported on every 8th iteration.
```

Kehitetyn ohjelman tarkempi toiminta on seuraavanlainen:

1. Jaetaan laskentakoneet käsittelemään omia korkeusmallejaan. Mikäli solmukoneiden muisti ei riitä korkeusmallin pitämiseen muistissa jaetaan korkeusmalli useamman laskentasolmun kesken.

2. Luodaan normaalijakautunut $\mathcal{N}(vec(\mathbf{X}_0), \mathbf{\Sigma}(\sigma^2, \phi))$ virhepinta, missä $\mathbf{X}_0$ on korkeusmallin korkeusmatriisi ja $\mathbf{\Sigma}(\sigma^2, \phi)$ on valitun semivariogrammin parametrien mukainen spatiaalinen korrelaatio. Oletetaan, että korkeusmallin matriisissa $\mathbf{X}_0$ on valmiina rittävän syvälle painettu jokiuomasto, joka ei muutu normaalijakautuneen kohinan ja kuoppien täytön vaikutuksesta.

3. Täytetään virhettä sisältävän korkeusmallin kuopat Wangin algoritmillä.

4. Lasketaan pisteet, jotka voivat virrata annettuihin valumapisteisiin. Asetetaan nämä pisteet ykkösiksi ja muut nolliksi

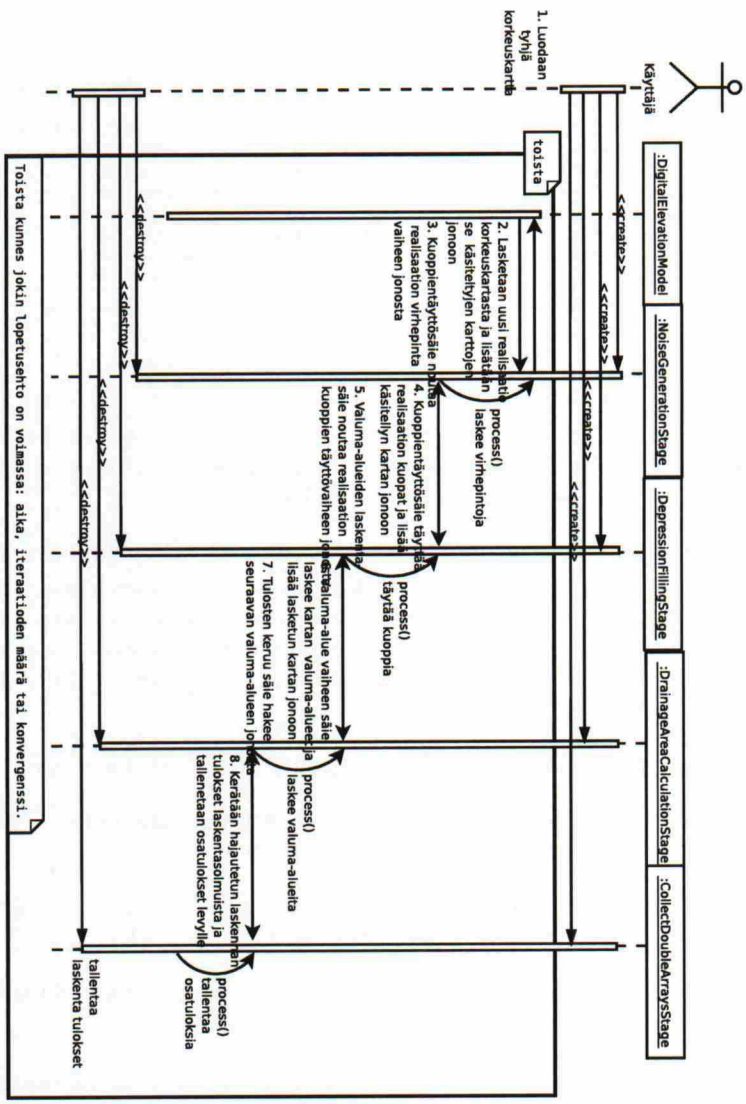
5. Kerätään lasketut valuma-alueet kaikilta solmukoneilta ja käytetään niitä uuden tarkemman keskiarvon laskemisessa $\mathbf{P}_i = \frac{1}{N} \sum \mathbf{D}_i$.

6. Estimoidaan haluttaessa konvergenssia luvussa kolme esitetyllä menetelmällä.

Vaiheita 1, 5 ja 6 ei työssä hajautettu. Loput vaiheet 2, 3 ja 4 lasketaan kehitettyjen hajautettujen algoritmien avulla.

5.2 Tekninen toteutus

Ohjelma toteutettiin pääasiassa Linux ja Unix ympäristön GNU [GCC] ohjelmointityökaluja käyttäen. Ohjelmointikieleksi ja ohjelmointitavaksi valittiin olio-orientoitunut C++ sille löytyvien, sovelluksessa tarvittavien, tehokkaiden C ja C++ kielisten apukirjastojen suuren määrän takia. Ohjelma suunniteltiin



Kuva 5.2: Liukuhihnan toiminta

toimimaan tyyppillisessä Linux/Unix ympäristössä ja ohjelmoitiin lähes kaikkien ominaisuuksiensa osalta kokonaan itse. Tällöin varsinaista ohjelmakoodia ja toteutusta testaavaa koodia kertyi yhteensä n. 25000 riviä.

Toteutetussa ohjelmassa valuma-alueanalyysin osalta toteutettiin lähes kaikista aiemmin kuvatuista valuma-alueiden laskentamenetelmistä sekä hajautetut että hajauttamattomat algoritmit. Näin algoritmien toiminan oikeellisuus voitiin vertailemalla varmistaa. Jensonin/Dominguen algoritmistä toteutettiin vain hajauttamaton toteutus. Kehitetetyn hajautetun kuoppientäyttöalgoritmin osalta algoritmien oikeellisuus tarkastettiin vertailemalla laskennan tuloksia Jensonin/Dominguen ja Wangin/Liun algoritmien antamien tulosten kanssa. Virhepinnan generoinnin osalta toteutettiin teorian mukainen prosessikonvoluutioon pohjautuva menetelmä, jossa Fourier-muunnoksen laskennan osalta käytettiin apuna ulkopuolista FFTW [FFTW] kirjastoa.

Hajautetun laskennan osalta ohjelmassa käytettiin MPI ohjelmointirajapintaa. Geodeettisella laitoksella ohjelma käytti LAM MPI kirjastoa [LAM] ja Tieteellisen laskennan klusterilaskenta ympäristössä MPICH-kirjastoa. Laskennassa tarvittavana, eräajoja koordinoivana klusterin resurssienhallintaohjelmana käytetiin Geodeettisella laitoksella työtä varten viritetyn kahden koneen miniklusterin osalta ilmaista Torque ohjelmaa [TRQUE]. Tieteellisen laskennan CSC:n laskentaympäristössä käytettiin Sun N1 Grid Engine [SGRID] ohjelmaa. Rinnakkaislaskennan osalta ohjelma kehitettiin toimimaan sekä POSIX Threads että GNU Pth kirjasto rajapintojen avulla. Näistä jälkimmäinen simuloi säikeitä itse ohjelman sisällä, minkä avulla pystyttiin kiertämään Tieteellisen laskennan ympäristössä vastaan tulleet ongelmat ohjelman ajamisessa.

Lopuksi tulosten tarkkuuden eli konvergenssin osalta johdetut monitorointimenetelmät toteuttiin hajauttamattomasti ja toteutuksen toimintaa testattiin normaalijakautuneella, nollakeskiarvoisella testiaineistolla. Toteutuksessa Fisherin $\mathcal{F}_{n,p}$-jakauman kertymäfunktion toteutuksen osalta käytettiin apuna GNU Scientific Library [GSL] kirjastoa. $\mathcal{F}_{n,p}$-jakauman kertymäfunktion laskenta osoittautui erittäin hankalaksi suurilla n:n ja p:n arvoilla, joilla lähes kaikki funktion laskentatavat divergoituivat nopeasti pois oikeasta arvostaan. GSL kirjastolla kertymäfunktion arvo laskettiin lopulta integroimalla numeerisesti Beta jakaumaa, jota apuna käyttäen saatiin laskettua $\mathcal{F}_{n,p}$ jakauman kertymäfunktio riittävällä tarkkuudella. Tätä poikkeusta lukuunottamatta laskennassa tarvitut matemaattiset menetelmät, mm. lineaarialgebran laskentamenetelmät, toteutettiin ja testattiin itse.

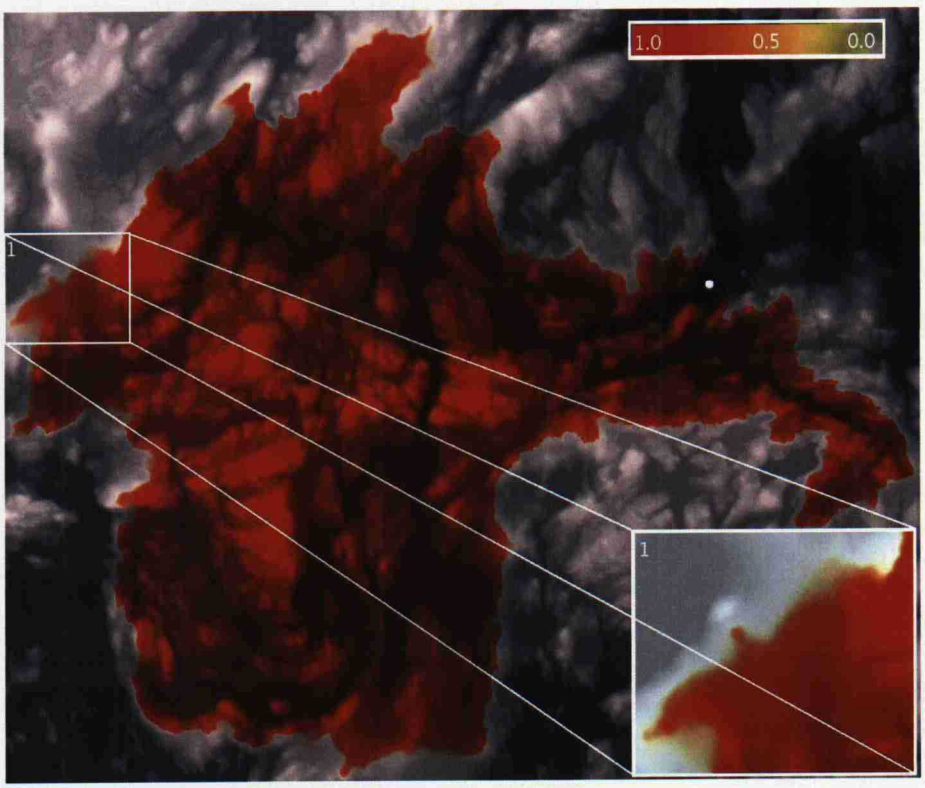
Luku 6

Tulokset

Toteutetun ohjelman ja algoritmien toimintaa testattiin aluksi keinotekoisella koeaineistolla. Näin varmistuttiin toteutettujen menetelmien oikeellisuudesta. Aluksi algoritmejä testattiin kokeilemalla, olivatko tulokset uskottavia ja testaamalla menetelmiä tapauksissa, joissa tarkka tai likimääräinen oikea tulos oli tiedossa. Esimerkiksi virhepintojen osalta laskettiin kokeellinen semivariogrammi, keskiarvo ja varianssi, sekä testattiin muuttujien normaalijakautuneisuutta. Tämän jälkeen varsinaisissa kokeissa testatuiksi asioiksi valittiin

- Wangin/Liun ja Jensonin/Dominguen kuoppientäyttöalgoritmien väliset nopeuserot,
- konvergenssianalyysimenetelmän toiminta ja sen antamat tulokset valuma-aluelaskennassa, sekä
- hajautetun laskentamenetelmän nopeus suhteessa hajauttamattomaan toteutukseen.

Testeissä käytettyjen laskentaympäristöjen tärkeimmät tekniset tiedot on koottu oheiseen taulukkoon. "Nopeus"-rivillä ilmoitetut lukemat ovat Linux käyttöjärjestelmän `/proc/cpuinfo` tiedoston mukaisia arvioita (BogoMIPS) koneen prosessorin nopeudesta (MIPS - millions of instructions per second).



Kuva 6.1: Esimerkki valuma-aluelaskennan tuloksesta

	PC-kone 1	**FGI:n klusteri**	**CSC:n klusteri**
Prossessorit	Athlon XP	Athlon XP [2]	HP Proliant Cluster
	1900+, 32 bit	UltraSparc IIi	Opteron 2.2 GHz
		32bit, 64bit	64bit, 512 CPUs
Nopeus	3170	3170 , 876	4390 per CPU
Muisti	1 GB RAM	1 GB, 512 MB	4 GB - 8 GB
Ohjelmistot	Linux 2.6	Linux 2.6	Linux 2.4, MPICH
		Torque,LAM	N1 Grid Engine
	PC-kone 2	**Kone 3**	
Prosessorit	Pentium 4	UltraSparc IIi Sabre	
	3.06 GHz, 32bit	64bit	
Nopeus	~5668[1]	876	
Muisti	1.5 GB RAM	512 MB	
Ohjelmistot	Win 2000 SP4	Linux 2.6	
	ARC/INFO 9.1		
	Matlab 6.5		

[1] SiSoft Sandra mittausohjelman tuloksista laskettu, lineaarisesti ekstrapoloitu arvio
[2] sama ensimmäisessä sarakkeessa kuvattu PC-kone 1

Kuoppientäyttöalgoritmien nopeuseroja vertailtiin mittaamalla laskentamenetelmien nopeutta muuttaen vain virhepinnan parametrejä. Konvergenssimenetelmää taas testattiin sekä keinotekoisella että oikealla aineistolla, jolloin saatiin arvio laskentamenetelmän antamasta tarkkuudesta. Laskennan hajautuksen osalta vertailtiin Wangin/Liun valuma-alue algoritmin toimintaa ilman hajautusta ja hajautuksen kanssa.

Kaikissa oikeaa korkeusmalliaineistoa käyttävissä testeissä laskentamenetelmien toimintaa kokeiltiin Hämjoen ympäristöstä lasketuista korkeusmalleista. Alkuperäinen testeissä käytetty Hämjoen korkeusmalli oli 10 metrin hilakoon mukainen 1447 x 1198 kokoinen korkeusmalli, joka oli testejä varten skaalattu myös pienemmiksi 723 x 599 ja 361 x 299 kokoisiksi korkeusmalleiksi. Korkeusmallin alueena käytettiin suorakulmion muotoista aluetta, jonka yhtenäiskoordinaatiston mukaiset kulmapisteet ovat (p = 6705000, i = 3318500) ja (p = 6694500, i = 3330500). Korkeusmallin origo yhtenäiskoordinaateissa on näistä ensimmäisessä eli pisteessä (p = 6705000, i = 3318500). Valumapisteenä käytettiin kaikissa testeissä pistettä (p = 6709085, i = 3327017).

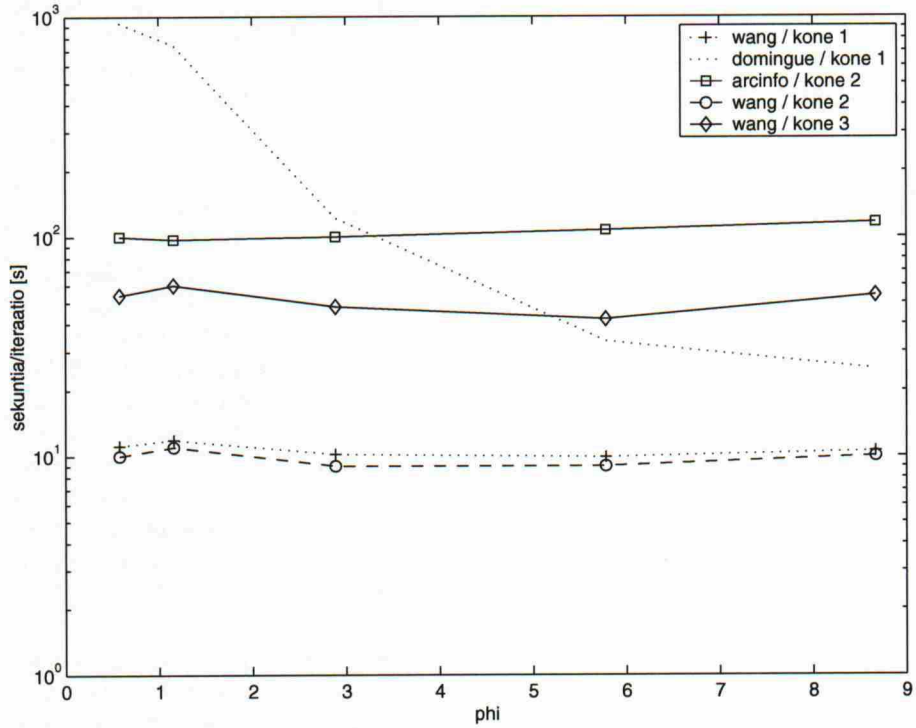
6.1 Kuoppientäyttöalgoritmien toiminta

Teoreettisen tarkastelun pohjalta Jensonin/Dominguen algoritmin laskentanopeuden tulisi olla voimakkaasti riippuvainen kuoppien määrästä. Wangin/Liun algoritmin nopeuteen kuoppien määrällä ei taas pitäisi olla lähes lainkaan vaikutusta. Tätä teoreettisen tarkastelun mukaista tulosta laskentanopeuteen haluttiin testata myös käytännössä. Algoritmin skaalautuvuus paitsi korkeusmallin varsinaisen koon, niin myös korkeusmallissa olevien kuoppien mukaan on tärkeää. Monte Carlo simuloinnissa luotavat korkeusmallit sisältävät tavallisesti käytetyillä, hyvin paikallisesti korreloituneilla virhepinnoilla, paljon pieniä kuoppia.

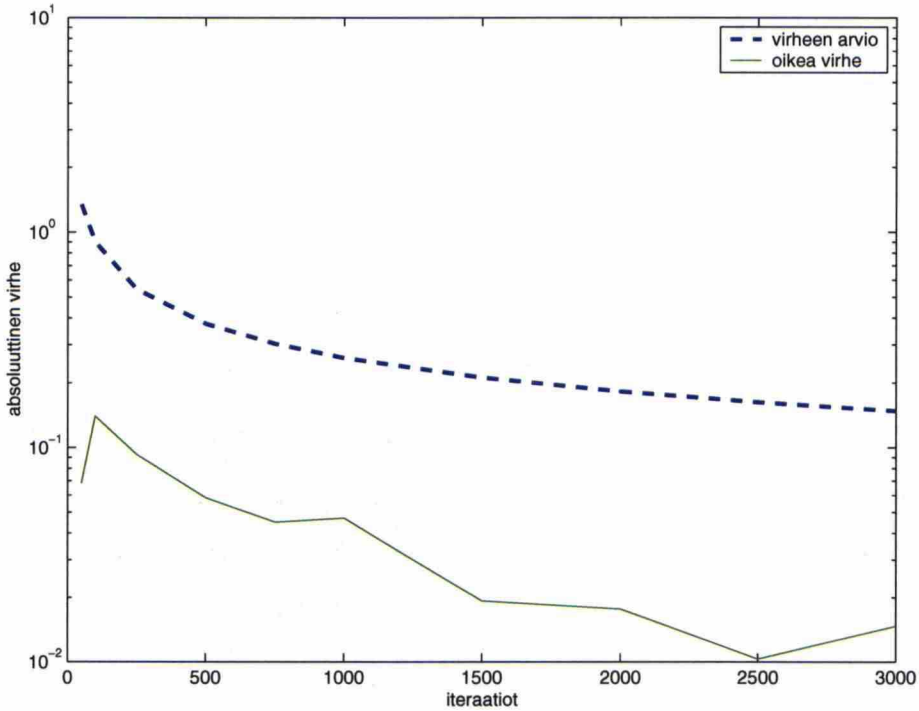
Laskentamenetelmien nopeutta virhepinnan kuoppaisuuden suhteen vertailtiin ajamalla algoritmejä eri virheparametrin arvoilla. Testissä käytetyn gaussisen semivariogrammin parametreiksi valittiin $\sigma^2 = 1$, $\phi = \{(0,57735,)\ 1,15607,\ 2,89017,\ 5,78035,\ 8,67052\}$.

Kuoppientäyttömenetelmien osalta testit tehtiin aiemmassa taulukossa esitellyillä Athlon XP, Pentium 4 ja UltraSparc koneilla ja ohjelmilla, joissa kaikissa ajettiin hajauttamatonta valuma-alue laskentaa. Windows 2000 ympäristössä laskenta tehtiin skriptillä, joka loi virhepinnat Matlabin toteutuksella ja laski valuma-alueet ARC/INFO 9.1 Workstation ohjelman algoritmeillä, jotka perustuvat Jensonin/Dominguen kuoppientäyttömenetelmään.

Tuloksista (Kuva 6.2) nähdään työssä toteutetun Wangin/Liun algoritmiin perustuvan kuoppientäyttömenetelmän olevan selvästi Jensonin/Dominguen menetelmää nopeampi. Myös ARC/INFO:n kuoppientäyttö algoritmi on selvästi Wangin/Liun algorimiä hitaampi. Hajautettavaksi valittu laskentamenetelmä on hajauttamattomana huomattavasti muita nopeampi. Kuten odotettua, virhepinnan parametrit eivät erityisesti vaikuttaneet Wangin/Liun laskentamenetelmän nopeuteen, kun taas suurilla ϕ:n arvoilla Jensonin/Dominguen menetelmässä tapahtui todella suurta hidastumista. Näin ei kuitenkaan käynyt ARC/INFO 9.1 Workstation ohjelman algoritmeille, joiden algoritmejä ilmeisesti optimoitu toimimaan Jensonin/Dominguen alkuperäistä perusalgoritmia nopeammin. Tarkat numeeriset tulokset mittauksista löytyvät työn liitteistä.



Kuva 6.2: Laskennan nopeus eri kuoppientäyttöalgoritmien suhteen



Kuva 6.3: Virheen arvio ja oikea virhe, kun suurin ominaisarvo on tiedossa

6.2 Tulosten tarkkuuden arviointi

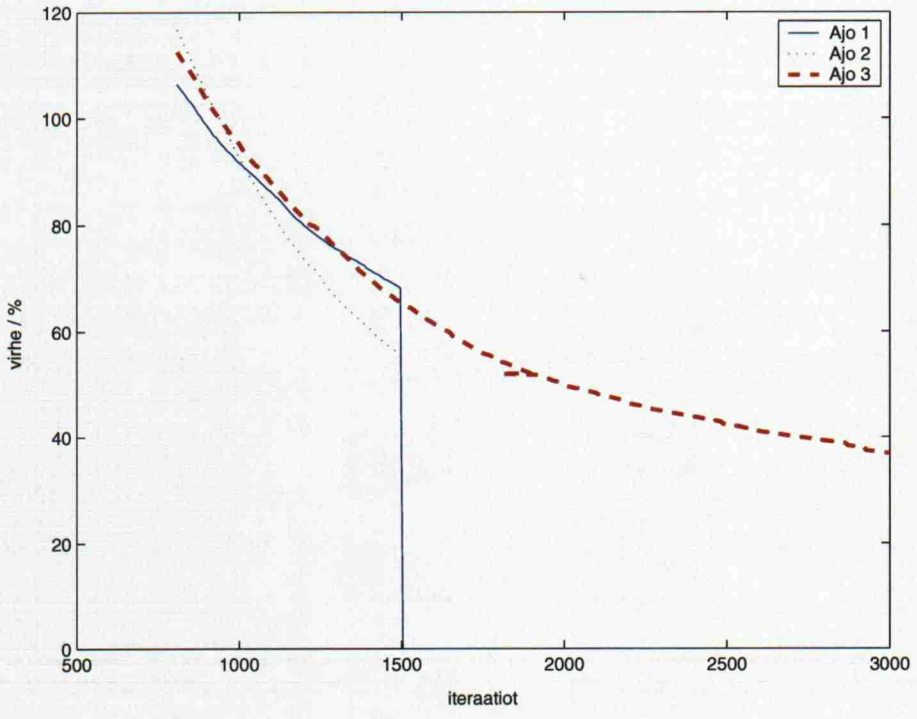
Kehitettyä laskentamenetelmää kokeiltiin aluksi laskemalla satunnaisia korrelaatioita sisältävälle normaalijakautuneelle, nollakeskiarvoisien muuttujien $\mathbf{y}$ keskiarvon virheen yläraja, kun kovarianssimatriisin suurimmaksi ominaisarvoksi oli asetettu $\lambda_{max} = 1$.

$$\mathbf{y} = \mathbf{Bx},\ \mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),\ \mathbf{A} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}, \mathbf{I})$$

$$\mathbf{B} = \mathbf{A}/\sqrt{max(eig(\mathbf{AA}^T))},\ \mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{BB}^T)$$

Tällöin suurinta ominaisarvoa ei jouduttu laskennan aikana arvioimaan. Menetelmää testattiin ajamalla se näin luoduille satunnaismuuttujille 1000 kertaa, jolloin oikea virhe ei ylittänyt 500 näytteen keskiarvoille laskettua ylärajaa kertaakaan. Tämän perusteella luottamusväli on $\geq 99,9\%$luottamusväli. Varsinaisissa testeissä keskiarvon oikea virhe oli n. 10 kertaa laskettua virhearvioita pienempi (Kuva 6.3).

Tämän jälkeen laskettiin virheen yläraja arviot aiemmin esitellylle Hämjoen korkeusmallin valuma-aluelaskennalle. Valuma-alue laskettiin kahdesti käyttäen 1500 iteraatiota eli korkeusmallin realisaatiota (Ajot 1 ja 2) ja kerran käyttäen 3000 iteraatiota (Ajo 3). Lasketun yläraja-arvion perusteella nähdään (Kuva 6.4) laskentamenetelmän toimivan järkevästi myös oikealla aineistolla. Toisaalta valuma-aluekartoissa $\mathbf{P}$ ei 1500 ja 3000 iteraation välillä ollut suurta eroa, joten

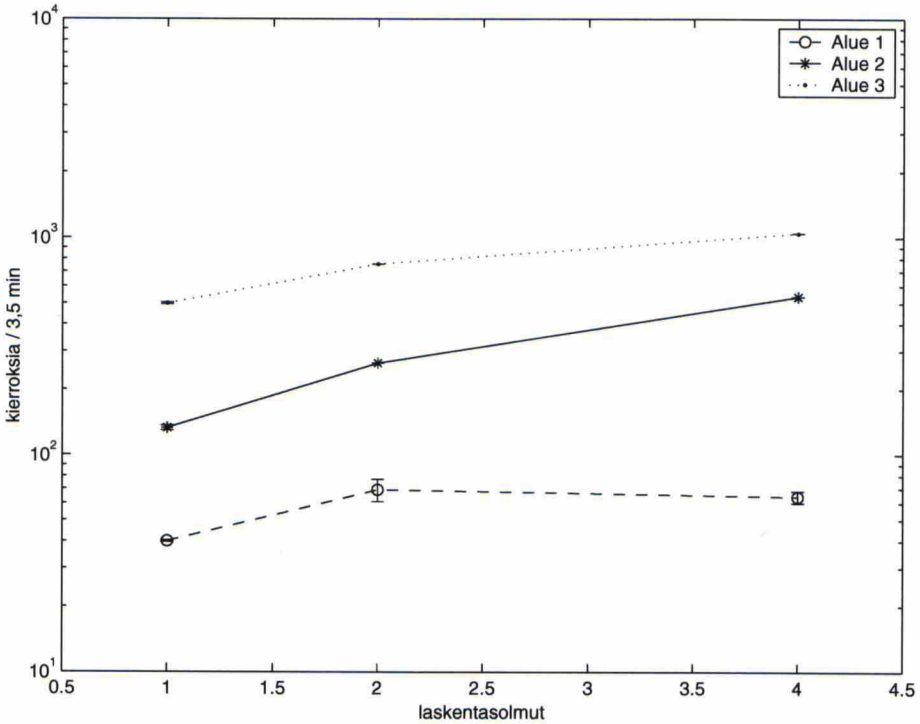


Kuva 6.4: Virheen 95% luottamusvälin yläraja Hämjoen korkeusalueelle

lasketun virheylärajan täytyy olla hyvin karkea, oikean virheen ja ilmeisesti myös tarkan 95% luottamusvälirajan ollessa huomattavasti matalampi. Menetelmässä hyödynnetty pääkomponenttianalyysi toimi sen sijaan hyvin. Käytetty Hämjoen 1447 x 1198 kokoisen valuma-aluekartan varianssi saatiin usein tiputettua alle kymmeneen muuttujaan ja aina alla sataan.

Laskennan konvergenssin arviointimenetelmää valuma-alue laskennassa olisi hyvä pyrkiä kehittämään tarkemmaksi. Yksi mahdollinen tapa tähän olisi pyrkiä huomioimaan keskiarvon normaalijakaumasta poikkeava, rajallinen alueväli $[0, 1]$. Kokeellisesti jo tarkkoja tuloksia antavan 1000 iteraation kohdalla virheraja on vasta luokkaa 100%, mikä tiedetään varmasti jo laskennan alussa. Kehitettyä konvergenssin arviointimenetelmää voidaan kuitenkin käyttää jo nyt, mikäli tulosten oikeellisuudesta halutaan olla lähes täysin varmoja ja käytössä on riittävästi laskentaresursseja riittävän suuren iteraatiomäärän laskemiseksi. Toisaalta, luvussa 3 määritellyn likimääräisen keskivirheen m_e kannalta tuloset antavat tilanteesta huomattavasti positiivisemman kuvan. Noin 1000 x 1000 kokoiselle alueelle lasketun valuma-alueen koko on samaa suuruusluokkaa koko korkeusmallin kanssa, minkä seurauksena keskivirhe on noin 1000 kertaa pienempi. Tällöin likimääräinen laskennallinen keskivirhe tuhannen iteraation paikkeilla olisi vain luokkaa $0,1\%$.

Kokeiden perusteella hajauttamattoman laskentamenetelmän mitatut ajoajat testikoneella (PC-kone 1) olivat 4,2 tuntia (1500 iteraatiota) ja 8,6 tuntia (3000 iteraatiota), joten ohjelmanajon pysäyttäminen tyypillisten konvergenssivaatimusten perusteella vaatisi ilmeisesti satoja tunteja laskenta-aikaa.

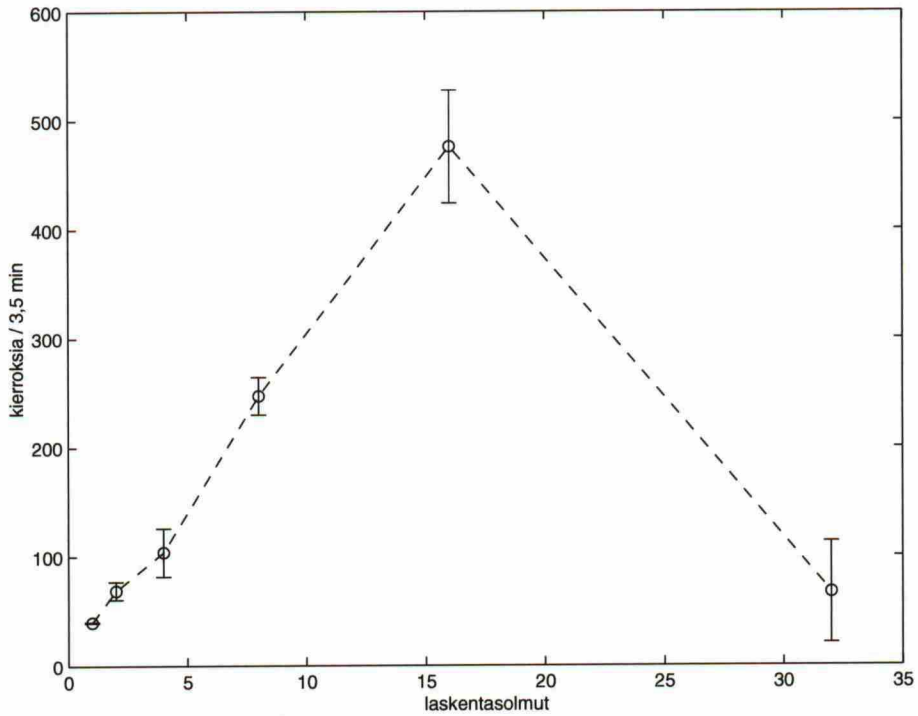


Kuva 6.5: Korkeusmallin koon vaikutus laskentanopeuteen

6.3 Hajautettu laskentamenetelmä

Hajautetun laskentaohjelman osalta ohjelma toimivuudesta varmistuttiin vertailemalla hajautetun ja hajauttamattoman Liun/Wangin kuoppientäyttöalgoritmin antamia tuloksia keskenään. Algoritmien nopeuksia testattiin aiemmin esitetyissä laskentaympäristöissä. Testiaineistona käytettiin Hämjoen valuma-aluetta ja laskennasta mitattiin 3,5 minuutissa laskettujen valuma-alue realisaatioiden määriä. Ohjelmia ajettiin niin, että konvergenssiarviointiin liittyvää laskentaa ei oltu kytketty päälle, jolloin lasketut ajat mittaavat yksinomaan varsinaisen valuma-alue laskennan nopeutta. Testeissä ohjelmaa ajettiin kolmella eri korkeusmallin korkeusmallien koolla. Erikokoisiksi skaalaatun korkeusmallien koot olivat 1447×1198 (Alue 1), 723×599 (Alue 2) ja 361×299 (Alue 3). Näistä kahdessa jälkimmäisssä on $\frac{1}{4}$ ja $\frac{1}{16}$ alkuperäisen korkeusmallin pistemäärästä. Testeissä virhepinnan semivariogrammina käytettiin gaussista semivariogrammia parametreillä $\sigma^2_{1447} = 1,0$ ja $\phi_{1447} = 1,0$, joita skaalattiin korkeusmallin koon pienentyessä, $\sigma^2_n = \sigma^2_{1447}$ ja $\phi_n = \frac{n}{1447}\phi_{1447}$. Eri korkeusmalleilla ja virheparametreillä saatiin oheisen taulukon mukaiset tulokset. Tuloksia vastaavat mittaustulokset on nähtävissä kuvassa 6.5.

Korkeusmalli/#CPU	FGI (2)	CSC (1)	CSC (2)	CSC (4)
Alue 1	3 (N=1)	40 (N=3)	69 (N=5)	104 (N=4)
Alue 2	11 (N=1)	133 (N=3)	264 (N=1)	536 (N=1)
Alue 3	60 (N=1)	499 (N=2)	756 (N=1)	1044 (N=1)



Kuva 6.6: Laskennan skaalautuvuus CSC:n klusterissa

Lisäksi testattiin hajautettujen laskentamenetelmien skaalautuvuutta useammille laskentasolmuille. Testissä käytettiin jatkuvasti samaa aluetta 1 ja nopeusmittaukset tehtiin samoin kuin edellisessä testissä mittaamalla 3,5 minuutissa laskettujen valuma-alueiden määrää ja keskihajonta (Kuva 6.6).

Tulosten perusteella korkeusmallin pienentyminen neljäsosaan kasvattaa laskentanopeutta keskimäärin 2-4 kertaisesti. Laskenta skaalautuu suurempiin laskentasolmumääriin likimain samalla tavalla korkeusmallin koosta riippumatta. Mielenkiintoisesti Geodeettisen laitoksen minitestausklusteri nopeutui korkeusmallin alueen pienentyessä jopa CSC:n klusteria paremmin. Syynä tähän on luultavasti Opteronia tehottomattommat prosessorit, jotka nopeutuvat verrattaen enemmän korkeusmallin koon pienentyessä.

Jälkimmäisten mittaustulosten (alue 1) perusteella nähdään algoritmin nopeuden skaalautuvan lineaarisesti suuremmille laskentasolmujenmäärille. Poikkeuksena tähän ovat kierrosmäärät 32:lla laskentasolmulla. Syynä tähän on ilmeisesti laskennan alussa tapahtuvat aikaa vievät tiedonsiirto ja laskentaresurssien jakovaiheet. Kun laskentasolmujen määrä kasvaa 32:teen, ei itse valuma-alueiden laskentavaihetta ehditä aloittamaan. Havaintojen perusteella varsinaisen laskentavaiheen nopeus vaikutti kuitenkin olevan suhteessa samaa luokkaa kuin pienemmillä laskentasolmumäärillä. Mittaustuloksien keskihajonta oli melko suurta. Tämä johtuu todennäköisesti klusterissa yhtä aikaa ajettavien ohjelmien vaikutuksesta tiedonsiirtonopeuksiin, sekä laskentaympärisön prosessorien heterogeenisuudesta. CSC:n klusterissa yhdellä prosessorilla voi olla 1-4 itsenäistä laskentaydintä, minkä seurauksena laskenta on nopeampaa tapauksissa,

joissa käytetyt laskentasolmut ovat vain muutamassa fyysisesti erillisessä prosessorissa.

Sovittamalla lineaarinen $y = ax + b$ malli kuvan 6.6 mittauksiin ja jättämällä 32 laskentasolmun mittaustulokset huomioimatta ovat neliöllisen virheen minimoivat kertoimet $a = 29,45$ ja $b = 4,58$. Lukuunottamatta alun tiedonsiirto ja laskentaresurssien jakovaihetta laskentamenetelmät skaalautuvat siten kohtuullisesti suuremmillekin prosessorimäärille. Kun yhdellä laskentasolmulla saadaan valuma-alueita laskettua 40 iteraatiota (kierrosta) ja yhden laskentasolmun lisäys kasvattaa iteraatioiden määrää 29,45:llä, niin lisälaskentasolmut lisäävät laskentakapasiteettia 74% teholla.

Luku 7

Yhteenveto

Työssä kehitettiin ja toteutettiin hajautetut valuma-alueiden laskenta-algoritmit, joiden avulla korkeusmallien epävarmuuden ja virheiden vaikutukset valuma-alueeseen voidaan laskea aiempaa nopeammin. Ohjelman käyttäjä voi määritellä kumulatiivisesti gaussisten ja eksponentiaalisten semivariogrammien avulla virheiden korrelaation korkeusmallissa ja laskea valuma-alueet merkitsemiinsä valumapisteisiin. Koska kehitettyllä hajautetun laskennan menetelmällä voidaan valuma-alueen laskenta jakaa usealle koneelle, pystytään käytettävissä olevien laskentaresurssien rajoissa valuma-alueita laskemaan nopeasti myös hyvin suurille korkeusmalleille. Haluttaessa ohjelma pystyy myös laskemaan arvioita tuloksessa vielä olevasta virheestä.

Mitattujen tuloksien perusteella työssä asetetut tavoitteet saavutettiin. Kehitetty, uuteen Wangin ja Liun algoritmiin perustuva menetelmä on huomattavasti aiempaa laskentaa menetelmää tehokkaampi ja se toimii nopeasti helpoissa tapauksissa, joissa kukin hajautetun laskennan kone voidaan laittaa laskemaan omaa koko korkeusmallin kattavaa korkeusmallia.

Testien mukaiset virhearviot antavat hyvin pessimistisen arvion korkeusmallissa vielä olevasta virheestä, joka ilmeisesti voi olla kymmeniä tai satoja kertoja annettua arviota pienempi. Lasketut virhearviot ovat siten hyödyllisiä vain riittävän pienillä korkeusmalleilla, joilla korkeusmallien realisaatioita voidaan vielä laskea huomattavasti enemmän kuin tarkan luottamusvälin mukainen virheraja vaatisi. Tulevaisuudessa olisi hyödyllistä pyrkiä kehittämään luottamusvälin arviointimenetelmiä antamaan realistisempia tuloksia. Koska pääkomponenttimenetelmällä pystyttiin tiputtamaan konvergenssiarviossa tarvittavien muuttujien määrä hyvin pieneksi, avaa tämä lähestymistapa mahdollisuuden soveltaa konvergenssinarviointiongelmaan kehittyneempiä, laskennallisen raskautensa takia vain matalaulotteisille aineistoille sopivia laskentamenetelmiä.

Hajautetun laskennan algoritmien osalta kehitetyn menetelmän pullonkaulaksi muodostui tulosten keruuvaiheessa tehtävä synkronointi kaikkien laskentasolmujen kesken. Tulosten keräysvaihe pakottaa laskennan lopulta tapahtumaan valuma-alueita laskevan solmujoukon hitaimman solmun mukaan.

Laskennan liukuhihnoitus ja rinnakkaistus itse laskentasolmussa auttaa jonkin tähän ongelmaan, mutta ei poista sitä seikkaa, että algoritmin jatkokehittelyssä tulosten keruuvaihe tulisi muuttaa toimimaan niin, että tuloksia keräävä solmukone ei kerää tuloksia aina kaikilta laskentasolmuilta. Vaikka puutte vaikuttaa laskentanopeuteen, se ei ole ongelma tyypillisessä klusterilaskentaympä-

ristössä, jossa kaikkien koneiden laskenta nopeus on lähes sama. Tätä auttaa myös ohjelman sisäinen heuristiikka, joka pyrkii asettamaan yhtä paljon laskentatehoa kuhunkin erikseen valuma-alueita laskevaan laskentasolmujoukkoon.

Kehitetyn ohjelman liukuhihnoitukseen perustuvan modulaarisen rakenteen takia ohjelmaa voidaan helposti muokata soveltumaan muuhunkin maanpinnan kaksiulotteista korkeusmallia käyttävään laskentaan. Yksi tälläinen sovelluskohde voisi olla esimerkiksi GSM tai muiden radio, TV tai WLAN tyyppisten verkkojen katvealueanalyysi radiomastojen tms. lähettimien optimaalisessa sijoittelussa.

Ohjelman muunneltavuus mahdollistaisi myös korkeusmallien näytteistämisen ja näytteiden jälkikäsittelyn parantamisen. Korkeusmallirealisaatioiden kuoppien poiston lisäksi laskentaan voitaisiin lisätä muutakin kvalitatiivisten virheiden korjausta. Toisaalta koko korkeusmallinäytteiden luontitapaa olisi myös hyvä pyrkiä kehittämään realistisempaan suuntaan pyrkimällä luomaan monimutkaisempia jakaumamalleja, joissa korkeusmallin kvalitatiivisia virheitä ei pääse muodostumaan. Tämä voi kuitenkin olla hyvin vaikeaa tai miltei mahdotonta, koska esimerkiksi jo kuopattomuuden vaatimus asettaa hyvin monimutkaisia riippuvuussuhteita korkeusmallin pisteiden välille.

Työn pohjana käytetyn korkeusmallin tilastollisen mallinnuksen suurimpana ongelman ovat jakauman parametrit, joiden arvioiminen käyttäjän antamista mittauksista olisi tarpeellista joko ohjelman osana tai erillisenä ohjelmana. Realististen parametrien valitseminen vaatii geostatistiikan asiantuntemusta ja tietoja käytetyssä aineistossa olevasta virheestä. Ohjelman käytön kannalta olisi hyvä pyrkiä kehittämään tälläistä parametrien estimointimenetelmää.

Kirjallisuutta

[Oks 05] Oksanen, J., Sarjakoski T. Error Propagation Analysis of DEM-based Drainage Basin Delineation. International Journal of Remote Sensing, Vol. 26, No. 14, 20 (July 2005), pp. 3085-3102.

[Oks 6] Oksanen, J. Sarjakoski T. Uncovering the Statistical and Spatial Characteristics of Fine Toposcale DEM Error. International Journal of Geographical Information Science. (painossa). 2006.

[Jen 88] Jenson S., Domingue J. Extracting Topographic Structure from Digital Elevation Data for Geographic Information System Analysis. Photogrammetric Engineering and Remote Sensing, Vol. 54, No. 11 (November 1989), pp. 1593-1600.

[Pla 01] Planchon, O., Darboux, F. A Fast, Simple and Versatile Algorithm to Fill The Depressions of Digital Elevation Models. Catena, Vol. 46 (2001), pp. 159-176.

[Wan 05] Wang L., Liu, H. Identification and Filling of Surface Depressions in Massive Digital Elevation Models for Hydrological Modelling. International Journal of Geographical Information Science, Vol. 20, No. 2 (February 2006), pp. 193-213.

[Swa 99] Swall, J. Non-Stationary Spatial Modelling Using A Process Convolution Approach. Ph.D. thesis. Duke University, Durham, USA. 1999.

[Ker 00] Kern II, J. Bayesian Process Convolution Approaches To Specifying Spatial Depedence Structure. Ph.D. thesis. Institute of Statistics and Decision Sciences. Duke University, Durham, USA. 2000.

[Gon 99] Gonzalez, R., Woods, R. Digital Image Processing. Prentice Hall, 2nd edition. ISBN 0201180758. 2002.

[Mit 98] Mitra, S. Digital Signal Processing - A Computer Based Approach. McGraw-Hill Series In Electrical and Computer Engineering. ISBN 0-07-042953-7. 1998

[Hay 96] Hayes, Monson H. Statistical Digital Signal Processing and Modelling. Wiley. ISBN 0471594318. 1996.

[Hay 98] Haykin, Simon. Neural Networks: A Comprehensive Foundation. Prentice Hall, 2nd edition. ISBN 0132733501. 1998.

[Ole 99] Olea, R. A. Geostatistics for Engineers and Earth Scientists. Kluwer Academic Publishers. ISBN 0-7923-8523-3. 1999.

[Gel 03] Gelman, A., Carlin, J., Stern, H., Rubin, D. Baysian Data Analysis. Chapman & Hall/CRC, 2nd edition. ISBN 158488388X. 2003.

[Ued 02] Ueda, N. Ghahramani, Z. Bayesian Model Search for Mixture Models Based on Optimizing Variational Bounds. Neural Networks, Vol. 15 (2002), pp. 1223-1241.

[Rob 04] Robert, Christian P., Casella, George. Monte Carlo Statistical Methods, 2nd edition. Springer Science+Business Media Inc. ISBN 0-387-21239-6. 2004.

[Gup 99] Gupta, A., Nagar, D. Matrix Variate Distributions. Chapman&Hall / CRC. ISBN 1-58488-046-5. 2000.

[Gol 83] Golub, Gene. Matrix Computations. North Oxford Academic Publishing Co. Ltd. ISBN 0-946536-00-7. 1983.

[And 99] Andrews, G. Foundations of Multithreaded, Parallel and Distributed Programming. Addison-Wesley. ISBN 0201357526. 1999.

[Box 58] Box, G., Muller, M. A Note on the Generation of Random Normal Deviates. The Annals of Mathematical Statistics, Vol. 29 (1958), pp. 610-611.

[Mar 00] Marsagalia, G., Tsang, W. The Ziggurat Method for Generating Random Variables. Journal of Statistical Software, Vol. 26 (2000), pp. 363-372.

[Pre 92] Press, W., Teukolsky, S., Vetterling, W., Flannery, B. Numerical Recipes in C - The Art of Scientific Computing. Cambridge University Press, 2nd edition. ISBN 0-521-43108-5. 1992. Numerical Recipes in C `http://www.library.cornell.edu/nr/bookcpdf.html`. Viitattu 23.1.2006.

[Knu 98] Knuth, D. Art of Computer Programming, Volume 3: Sorting and Searching. Addison-Wesley Professional, 2nd edition. ISBN 0201896850. 1998.

[Man 05] Mans, Bernand. Portable Distributed Priority Queues with MPI. Concurrency: Practice and Experience, Vol. 10, Issue 3 (March 1998), pp. 175-198.

Standardit, ohjelmistot ja ohjelmistokirjastot

[MPI95] MPI: A Message-Passing Interface Standard. Message Passing Interface Forum, Version 1.1. 1995.
MPI: A Message-Passing Interface Standard `http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html`. Viitattu 23.1.2006.

[LAM] LAM/MPI Parallel Computing. MPI standardin toteuttava kirjasto ja työkalut.
LAM/MPI Parallel Computing `http://www.lam-mpi.org/` Viitattu 23.1.2006.

[GCC] GNU Compiler Collection. C/C++ ohjelmointikielten kääntäjä ja muut GNU työkalut: Autoconf, GNU Make.
GCC Home Page `http://gcc.gnu.org/`
Autoconf `http://www.gnu.org/software/autoconf/`
GNU Make `http://www.gnu.org/software/make/`
Viitattu 23.1.2006.

[GSL] GNU Scientific Library. Laajahko apukirjasto matemaattiseen laskentaan.
GNU Scientific Library `http://www.gnu.org/software/gsl/`. Viitattu 26.1.2006.

[TRQUE] Torque. Klusterilaskentaympäristön eräajon resurssienhallintaohjelma.
Torque Resource Manager 2.0 `http://www.clusterresources.com/pages/products/torque-resource-manager.php`. Viitattu 23.1.2006.

[PTH] GNU Pth - GNU Portable Threads. Rinnakkaislaskennan kirjasto.
GNU Portable Threads `http://www.gnu.org/software/pth/`. Viitattu 1.2.2006.

[SGRID] Sun N1 Grid Engine. Eräajo ympäristön resurssienhallintaohjelma.
Sun N1 Grid Engine `http://www.sun.com/software/gridware/`. Viitattu 26.1.2006.

[FFTW] Fastest Fourier Transform in West. Nopea Fast Fourier Transform algoritmin toteuttava kirjasto.
FFTW Homepage `http://www.fftw.org/` Viitattu 23.1.2006.

[OCTA] GNU Octave. Korkean tason MATLAB tyylinen matemaattinen skriptauskieli ja laskenta ympäristö.
Octave Home Page `http://www.octave.org/` Viitattu 16.1.2006.

[HDRST] Spatial hydrology and statistical software. Linkkejä spatiaalisen hydrologian ohjelmistoihin ja tutkimusprojekteihin.
Spatial and Spatial Hydrology Software `http://www.spatialhydrology.com/software_hydrostat.html` Viitattu 27.1.2006.

[VBAYES] Variational-Bayes Repository. Bayesiläisen päättelyyn liittyvän laskentamenetelmän ohjelmia ja tutkimuspapereita, sekä linkkejä aiheeseen.
Variational-Bayes Repository `http://www.variational-bayes.org/` Viitattu 30.1.2006.

Liite A

Liitteet

A.1 Diskreetin konvoluutioteoreeman todistus

Prosessikonvoluutiota käsittelevässä kappaleessa on esitelty diskreettikonvoluutio $\mathbf{a} \star \mathbf{b}$, Fourier-muunnos $\mathcal{F}(\cdot)$, sekä konvoluutioteoreema, jonka yleinen muoto on

$$\mathcal{F}(\mathbf{a} \star \mathbf{b}) \propto \mathcal{F}(\mathbf{a})\mathcal{F}(\mathbf{b}).$$

Todistetaan nyt tämä hyvin tunnettu digitaalisen signaalinkäsittelyn tulos yksiulotteisessa tapauksessa eli vektoreilla. Koska konvoluutio ja Fourier-muunnos kohdistuvat erikseen kuhunkin ulottuvuuteen, voidaan seuraavaksi esiteltävät konvoluutio, Fourier-muunnos ja konvoluutioteoreema suoraviivaisesti yleistää luvun lopussa esitellyllä tavalla kahteen (matriisit) tai useampaan ulottuvuuteen. Olkoon $\mathbf{a}$ N-ulotteinen vektori ja $\mathbf{b}$ M-ulotteinen vektori, $\mathbf{a}[0..(N-1)]$, $\mathbf{b}[0..(M-1)]$. Vektoreiden diskreetti konvoluutio ja Fourier muunnos ovat määritelty seuraavasti [Mit 98].

$$(\mathbf{a} \star \mathbf{b})[m] = \sum_{n=max(0,m-(M-1))}^{min(N-1,m)} \mathbf{a}[n]\mathbf{b}[m-n]$$

missä $m \in [0, M+N-2]$ ja

$$\mathcal{F}(\mathbf{a})[u] \propto \sum_{n=0}^{N-1} \mathbf{a}[n] e^{-j(u\frac{2\pi}{N})n}$$

$$\mathcal{F}^{-1}(\mathbf{a})[n] \propto \sum_{u=0}^{N-1} \mathbf{a}[u] e^{j(n\frac{2\pi}{N})u}$$

Fourier-muunnos kaavoista on jätetty vakio skaalaustermi pois. Lineaarisen funktion skaalaus tulisi valita niin, että $\mathcal{F}^{-1}(\mathcal{F}(\mathbf{a})) = \mathbf{a}$. Tämä voidaan tehdä eri tavoilla, mutta näistä tavoista kaikki eivät johda eksaktiin konvoluutioteoreemaan $\mathcal{F}(\mathbf{a} \star \mathbf{b}) = \mathcal{F}(\mathbf{a})\mathcal{F}(\mathbf{b})$, joten tässä kappaleessa käytetään vain skaalamatonta Fourier-muunnosta ja todistetaan konvoluutioteoria ilman eksakteja skaalaustermejä.

Laskemalla konvoluution Fourier-muunnos saadaan johdettua konvoluutioteoreema nollilla laajennetuille diskreeteille signaaleille.

$$\mathcal{F}(\mathbf{a} \star \mathbf{b})[u] = \sum_{m=0}^{M+N-2} (\mathbf{a} \star \mathbf{b})[m] e^{-j(u\frac{2\pi}{M+N-1})m}$$

$$= \sum_{m=0}^{M+N-2} \Big(\sum_{n=max(0,m-(M-1))}^{min(N-1,m)} \mathbf{a}[n]\mathbf{b}[m-n] \Big) e^{-j(u\frac{2\pi}{M+N-1})m}$$

$$= \sum_{n=0}^{N-1} \sum_{m=n}^{n+M-1} \mathbf{a}[n] e^{-j(u\frac{2\pi}{M+N-1})n} \mathbf{b}[m-n] e^{-j(u\frac{2\pi}{M+N-1})(m-n)}$$

$$= \sum_{n=0}^{N-1} \left(\mathbf{a}[n] e^{-j(u\frac{2\pi}{M+N-1})n} \sum_{m=n}^{n+M-1} \mathbf{b}[m-n] e^{-j(u\frac{2\pi}{M+N-1})(m-n)} \right)$$

Laajentamalla nyt vektorit $\mathbf{a}$ ja $\mathbf{b}$ nollilla $N + M - 1$ pituisiksi vektoreiksi $\mathbf{a}_e$ ja $\mathbf{b}_e$, mikä ei vaikuta konvoluution arvoon, voidaan kaavan summatermit tulkita $N + M - 1$ pituisiksi diskreeteiksi Fourier-muunnoksiksi.

$$= \sum_{n=0}^{N+M-2} \left(\mathbf{a}_e[n] e^{-j(u\frac{2\pi}{M+N-1})n} \sum_{m=0}^{N+M-2} \mathbf{b}_e[m] e^{-j(u\frac{2\pi}{M+N-1})m} \right)$$

$$= \sum_{n=0}^{N+M-2} \left(\mathbf{a}_e[n] e^{-j(u\frac{2\pi}{M+N-1})n} \mathcal{F}(\mathbf{b}_e)[u] \right)$$

$$= \mathcal{F}(\mathbf{a}_e)[u]\mathcal{F}(\mathbf{b}_e)[u]$$

Tuloksesta saadaan laskukaava konvoluution laskemiseksi Fourier-muunnoksen avulla, joka voidaan toteuttaa nopeasti FFT-algoritmillä (Fast Fourier Transform).

$$\mathbf{a} \star \mathbf{b} \propto \mathcal{F}^{-1}\left[\mathcal{F}(\mathbf{a}_e)\mathcal{F}(\mathbf{b}_e)\right]$$

Tuloksen yleistäminen korkeampiin ulottuvuuksiin onnistuu, kun käytetään alla olevia kaavoja konvoluution ja Fourier-muunnoksen laskemiseen. Kaavojen kaksiulotteiset laajennokset on löydettävissä digitaalisen kuvankäsittelyn kirjallisuudesta[Gon 99] ja yleisemmät tulokset Fourier-muunnoksia käsittelevästä matemaattisesta kirjallisuudesta. Kaavoissa $\mathbf{A}$ ja $\mathbf{B}$ voivat olla esimerkiksi diskreettejä $N_1 \times N_2$ ja $M_1 \times M_2$ kokoisia matriiseja.

$$(\mathbf{A} \star \mathbf{B})[m_1, .., m_K] =$$

$$\sum_{n_1=max(0,m_1-(M_1-1))}^{min(N_1-1,m_1)} \cdots \sum_{n_K=max(0,m_K-(M_K-1))}^{min(N_K-1,m_K)} \mathbf{A}[n_1, .., n_K]\mathbf{B}[(m_1-n_1), .., (m_K-n_K)]$$

missä $m_i \in [0, M_i + N_i - 2]$ ja

$$\mathcal{F}(\mathbf{A})[u_1, .., u_K] \propto \sum_{n_1=0}^{N_1-1} .. \sum_{n_K=0}^{N_K-1} \mathbf{a}[n_1, .., n_K] e^{-j2\pi \sum_{i=1}^{K} \frac{u_i n_i}{N_i}}$$

$$\mathcal{F}^{-1}(\mathbf{A})[n_1, .., n_K] \propto \sum_{n_1=0}^{N_1-1} .. \sum_{u_K=0}^{N_K-1} \mathbf{a}[u_1, .., u_K] e^{j2\pi \sum_{i=1}^{K} \frac{u_i n_i}{N_i}}$$

A.2 Optimaalinen konvoloitavan virhepinnan jako

Todistetaan aiemmin annettu, konvoluution mukainen minimitiedon siirtomäärää koskeva tulos. Olkooon korkeusmallin koko $W \times H$, suotimen koko $(2X + 1) \times (2Y + 1)$ ja olkoon jokaisessa solmukoneessa käytettävissä olevaa muistia M yksikköä. Olkoon korkeusmalli paloiteltu K^2:teen, yhtä suureen $\frac{W}{K} \times \frac{H}{K}$ kokoiseen, suorakaiteen muotoiseen alueeseen. Tällöin korkeusmallit osa-alueet, joille on olemassa kaikki 8 naapuripalaa, vastaanottavat tietoa muista laskentasolmuista $4XY + 2XH/K + 2YW/K$ pistettä. Nämä pisteet ovat laskenta-alueen naapuripisteitä, jotka tarvitaan, jotta $X \times Y$ kokoisen suodinmaskin mukainen konvoluutio voidaan laskea jokaisessa laskentasolmun pisteessä. Olettamalla kaikilla laskentasolmuilla olevan 8 naapuria ja vähentämällä alueen reunapalojen ylimääräiset alueet voidaan tiedonsiirronmäärä laskea koko K^2 aluetta sisältävälle jaolle.

$$T(K) = 4XYK^2 + 2XHK + 2YWK - (2WY + 2HX + 4XY)$$

Kaava saa minimiarvonsa aina kun $K = 1$, joten on vielä tarpeen rajoittaa yhden solmukoneen muistinkäyttöä $HW = K^2M$.

$$T(K) = 2\tfrac{XM}{W}K^3 + (4XY - 2\tfrac{MX}{W})K^2 + 2YWK - (2WY + 4XY)$$

$$\tfrac{\partial T(K)}{\partial K} = 3\tfrac{XM}{W}K^2 + 2(2XY - \tfrac{MX}{W})K + YW = 0$$

Sijoittamalla kaavaan $\frac{\partial T(K)}{\partial K} = 0$, $HW = K^2M$ saadaan

$$3XH + 4XYK - 2X\tfrac{H}{K} + YW = 0$$

$$4XYK^2 + (3XH + YW)K - 2XH = 0$$

Sijoittamalla saatu tulos takaisin kaavaan $T(K)$ saadaan kaava optimaalisten ratkaisujen tiedonsiirtomäärälle $T(K) = (YW - XH)K - 4XY - 2WY = O(max(H, W)K)$. Yhtälöstä $\frac{\partial T(K)}{\partial K} = 0$ voidaan myös ratkaista reaaliarvoinen optimiarvo K:lle, jonka ympäristöstä voidaan etsiä hyviä yhtälön kokonaislukuratkaisuja.

$$K = \tfrac{1}{3M}\left(M - 6YW \pm \sqrt{M^2 - (4WX + 3W)M + 4X^2W^2}\right)$$

A.3 Ulottuvuuksien redusointi

Ohessa on C++ lähdekoodi, joka etsii alivaruuden, joka sisältää $p\%$ koko datassa olevasta varianssista.

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include <new>


/*
 * Heuristically estimates linear p% variance
 * dimensions (subspace) from data. Method
 * uses approximative PCA solution (~ uncorrelated
 * dimensions) to calculate such subspace (in practice
 * data between solved dimensions may correlate quite much)
 *
 * In any case, this method always calculates at least
 * the first five dimensions.
 *
 * Code is based on approximative PCA calculation
 * method (aproxeigs - modified Hebbian eigenfilter
 * (uses deflate method from FastICA)).
 *
 * Parameters:
 *
 * plimit   - p% percent stopping criteria
 * N        - number of data points
 * vdim     - number of dimensions in data
 * v        - data v[j,i] = v[j*vdim + i]
 * w        - subspace vectors (allocated with malloc() by the routine)
 *            (i:th vector: w[i*vdim + (0..vdim-1)] (0 <= i < wnum))
 * wnum     - number of subspace vectors estimated (subspace dimensions)
 * wvar     - amount of variance in estimated subspace
 * e        - vector to store solved eigenvalues
 *            (allocated with malloc() by the routine)
 * zeromean - does input data (v) has zero mean
 *
 * Returns total variance estimated to be in data or negative
 * value if there was an error.
 *
 * If call is succesful (and w != 0) then w contains
 * vectors of subspace. It is resposibility of caller
 * to free(w) those vectors (free(e) too).
 *
 * in pratice it could be also good idea to reduce
 * dimensions down to given limit so things are
 * computable but results are bad if subspace
 * covers only small proportion of all variance.
 *
 * This implementation is based on my earlier eigenfilter code.
 * Tomas Ukkonen
 *
 */
double approxvarspace(const double plimit, const unsigned long N,
      const unsigned long vdim, double* v, double*& w,
      unsigned long& wnum, double& wvar, double*& e,
      bool zeromean = true)
{
  // illegal parameter values
  if(plimit <= 0.0 || plimit >= 1.0) return -1.0;
  if(N <= 0 || vdim <= 0) return -1.0;


  double *w_old = 0, *s = 0, *d = 0;
  w = 0; e = 0;
```

```
  w     = (double*)malloc(sizeof(double)*vdim);
  e     = (double*)malloc(sizeof(double));

  w_old = (double*)malloc(sizeof(double)*vdim);
  s     = (double*)malloc(sizeof(double)*vdim);
  d     = (double*)malloc(sizeof(double)*vdim*N);


  if(w == 0 || w_old == 0 || s == 0 || d == 0 || e == 0){
    // memory allocation(s) failed

    if(w) free(w);
    if(w_old) free(w_old);
    if(s) free(s);
    if(d) free(d);
    if(e) free(e);

    w = 0; e = 0;
    wnum = 0;

    return -1.0;
  }


  // local copy of data: d
  memcpy(d, v, sizeof(double)*N*vdim);


  if(!zeromean){ // removes mean from data
    double* mean = w_old;
    memset(mean, 0, sizeof(double)*vdim);
    double s = 1.0 / ((double)N - 1.0);

    for(unsigned long n=0;n<N;n++)
      for(unsigned long j=0;j<vdim;j++)
mean[j] += s * d[n*vdim + j];

    for(unsigned long n=0;n<N;n++)
      for(unsigned long j=0;j<vdim;j++)
d[n*vdim + j] -= mean[j];
  }


  // calculates total variance from (zero mean) data
  double totalvar = 0.0;
  {
    double scale = 1.0/((double)N - 1.0);

    for(unsigned long i=0;i<(N*vdim);i++)
      totalvar += scale*(d[i]*d[i]);
  }


  const double varlimit = totalvar*plimit;
  wvar = 0.0; // amount of variance in already estimated subspace
  wnum = 0;   // number of estimated dimensions

  unsigned long index = 0;
  const double TOLERANCE = 0.000001;
  const unsigned long MAXITERS = 200;

  while(wvar < varlimit && index < vdim){

    // initialization of subspace direction
    // (random unit length vectors)

    {
      double* _temp = (double*)realloc(w, sizeof(double)*(index+1)*vdim);
      if(_temp == 0){
free(w); free(s);
free(d); free(w_old);

w = 0; wnum = 0;
return -1.0;
```

```
      }

      w = _temp;

      _temp = (double*)realloc(e, sizeof(double)*(index+1));
      if(_temp == 0){
free(w); free(s); free(e);
free(d); free(w_old);

w = 0; e = 0;
wnum = 0;
return -1.0;
      }

      e = _temp;
    }


    double wlen = 0.0;
    for(unsigned long i=0;i<vdim;i++){
      w[index*vdim + i] = rand()/((double)RAND_MAX) - 0.5;
      wlen += w[index*vdim + i]*w[index*vdim + i];
    }

    wlen = 1.0 / sqrt(wlen);

    for(unsigned long i=0;i<vdim;i++)
      w[index*vdim + i] *= wlen;


    bool convergence = false;
    unsigned long iters=0;

    while(!convergence){
      // keeps solving w[index] eigenvector
      memcpy(w_old, &(w[index*vdim]), sizeof(double)*vdim);

      for(unsigned long i=0;i<N;i++){
double n = 1.0/((double)(i+1));

double y = 0.0;

// y = (d[i]*w[index])[0];
for(unsigned long k=0;k<vdim;k++)
  y += d[i*vdim + k]*w[index*vdim + k];

// w[index] += n*(y*d[i] - y*y*w[index]);
double wlen = 0.0;
for(unsigned long k=0;k<vdim;k++){
  w[index*vdim + k] += n*y*(d[i*vdim + k] - y*w[index*vdim + k]);
  wlen += w[index*vdim + k]*w[index*vdim + k];
}

// w[index].normalize();
wlen = 1.0/sqrt(wlen);

for(unsigned long k=0;k<vdim;k++)
  w[index*vdim + k] *= wlen;
      }

      // orthonormalizes w[index] against
      // already solved w[0...(index-1)] basis

      // s.zero();
      memset(s, 0, sizeof(double)*vdim);

      for(unsigned long j=0;j<index;j++){
// s += (w[j]*w[index])*w[j];

double ww = 0.0;
for(unsigned long k=0;k<vdim;k++)
  ww += w[j*vdim + k]*w[index*vdim + k];

for(unsigned long k=0;k<vdim;k++)
```

```
  s[k] += ww*w[j*vdim + k];
      }

      // w[index] -= s;
      // w[index].normalize();
      double wlen = 0.0;
      for(unsigned long k=0;k<vdim;k++){
w[index*vdim + k] -= s[k];
wlen += w[index*vdim + k]*w[index*vdim + k];
      }

      wlen = 1.0/sqrt(wlen);
      for(unsigned long k=0;k<vdim;k++)
w[index*vdim + k] *= wlen;

      // checks for convergence
      // tmp = 1.0f - abs(w_old * w[index])[0];

      double tmp = 0.0;
      for(unsigned long k=0;k<vdim;k++)
tmp += w_old[k]*w[index*vdim + k];

      tmp = 1.0 - fabs(tmp);

      if(tmp < TOLERANCE)
convergence = true;

      iters++;
      if(iters >= MAXITERS)
break;
    }


    /*
     * removes effect of w[index] vector from data
     * and calculates how much variance this direction
     * collects from data
     */
    e[index] = 0.0;

    for(unsigned long i=0;i<N;i++){
      // d[i] -= ((d[i]*w[index])[0])*w[index];

      double innerproduct = 0.0;
      for(unsigned long k=0;k<vdim;k++)
innerproduct += d[i*vdim + k]*w[index*vdim + k];

      for(unsigned long k=0;k<vdim;k++)
d[i*vdim + k] -= innerproduct*w[index*vdim + k];

      e[index] += (innerproduct*innerproduct)/((double)N);
    }

    wvar += e[index];
    index++;
  }

  wnum = index;

  ////////////////////////////////////////////////
  // frees memory

  free(w_old);
  free(s);
  free(d);

  if(wnum == vdim)
    wvar = totalvar;

  // vectors of subspace w isn't free'ed
  // caller must free them with free().

  return totalvar;
}
```

A.4 Mittaustulokset

Tässä kappaleessa listataan kuvissa käytettyjä mittaustuloksia.

A.4.1 Hajautetun laskentamenetelmän mittaukset

Hajautetun algoritmin toimintanopeutta mitattiin ajamalla algoritmejä eri kokoisilla korkeusmalleilla ja vertailemalla saatuja laskenta-aikoja.

```
Source DEM file 'dd.bin' : 1447 x 1198
Source noise parameters
Gaussian correlation model
Variance (sigma^2) = 1, phi = 1.

Drainage point was (1027,466).
Area was Hamjoki drainage basin.
Convergence estimation calculations were not done.

Downscaled DEM files were: 723 x 599, 361 x 299.
For which phi parameters were 1/2 and 1/4.
And startpoints were (513, 233), (256,116).

Number of iterations computed in 3 minutes and 30 seconds
were measured. If exact execution time was longer (FGI cluster),
results were scaled accordingly. In case of
multiple measurements (N>1), mean value was used.

Execution times on CSC's cluster can be off 10-15 seconds and
exeuction speeds were clearly affected by other load in the cluster.

1447 x 1198 sized maps were large enough to fit the whole memory
so no map dividing was needed. CSC's CPUs had multiple cores but
algorithm was executed to use only one core per machine.
Execution was single threaded and multiple threads were simulated
with PTH library. CSC's computing environment (C+MPI) didn't allow
execution of multiple threads per process/node.
```

Algorithm / Machine	Map size 1447x1198	723x599	361x299
demanalysis Athlon XP [1]	22/N=1	82/N=1	431/N=1
mpidemanalysis, FGI cluster [2]	3/N=1	11/N=1	60/N=1
mpidemanalysis CSC cluster [3] 1 CPU (1x1 core)	40/N=3	133/N=5	499/N=2
mpidemanalysis CSC cluster [3] 2 CPUs (2x1 core)	69/N=5	264/N=1	756/N=1
mpidemanalysis CSC cluster [3] 4 CPUs (4x1 core)	104/N=4	536/N=1	1044/N=1

Scalability test

Mean number of iterations in 3 minutes and 30 seconds
on CSC's cluster. (N = number of samples)

Map size/CPUs	1	2	4	8	16	32
1447 x 1198 [3]	40/N=3	69/N=5	104/N=4	247/N=3	476/N=2	67/N=3

```
Machine / software information

[1] demanalysis (wang's algorithm)
Linux 2.6.12-1.1380_FC3 #1 i386 GNU/Linux AMD
Athlon XP 1900+ CPU (1.6 GHz) 1 GB RAM

[2] mpidemanalysis (distributed wang's algorithm)
FGI minicluster (2 machines): Athlon XP [1] and
UltraSparc IIi Sabre (64bit) 512 MB RAM

[3] mpidemanalysis (distributed wang's algorithm)
HP Proliant Cluster (sepeli.csc.fi)
2 CPU x Opteron 2.2 GHZ (SMP, 1 core)  4/8 GB RAM (~50% of nodes),
2 CPU x Opteron 2.2 GHZ (SMP, 2 cores) 4/8 GB RAM (~50% of nodes),
512 CPUs, 768 processor cores
```

A.4.2 Kuoppientäyttöalgoritmien ajoajat

Oheiset tulokset antavat eksaktit mittaustulokset eri algoritmien toimintanopeuksista, eri virhepinnan parametreillä.

```
DEM 'dd.bin' : 1447 x 1198
Variance (sigma^2) = 1
Gaussian correlation model/semivariogram

Dranage point was (1027,466).
Area was Hamjoki drainage basin.
Convergence estimation calculations were not done.

Measurement method with experiments [1] and [2] were
RDTSC x86 assembly command based timing
(accuracy: ~ 1/1.6 GHz sec/iter).
Measurement method with experiment [3], [4] and [5] was
based on system clock (accuracy ~0.5 sec/iter).
```

Parametrit (phi, iterations, #)	Total	Mean sec/iter	Std.Dev. sec/iter
0.57735, 100, [1]	18 min	11.1591	0.102094
1.15607, 100, [1]	19 min	11.8572	0.115461
2.89017, 100, [1]	16 min	10.1882	0.101709
5.78035, 100, [1]	16 min	9.8900	0.103126
8.67052, 100, [1]	17 min	10.5162	0.0990754
0.57735, 10, [2]	155 min	934.8	25.1248
1.15607, 10, [2]	124 min	744.398	63.8731
2.89017, 10, [2]	20 min	121.695	9.13081
5.78034, 10, [2]	5 min	33.3318	0.963961
8.67052, 10, [2]	4 min	24.9955	0.724911
0.57735, 10, [3]	N/A	100	N/A
1.15607, 10, [3]	N/A	97	N/A
2.89017, 10, [3]	N/A	100	N/A
5.78034, 10, [3]	N/A	107	N/A
8.67052, 10, [3]	N/A	116	N/A
0.57735, 10, [4]	N/A	10	N/A
1.15607, 10, [4]	N/A	11	N/A
2.89017, 10, [4]	N/A	9	N/A
5.78034, 10, [4]	N/A	9	N/A
8.67052, 10, [4]	N/A	10	N/A

```
0.57735,   10, [5]      9 min    54            N/A
1.15607,   10, [5]     10 min    60            N/A
2.89017,   10, [5]      8 min    48            N/A
5.78034,   10, [5]      7 min    42            N/A
8.67052,   10, [5]      9 min    54            N/A

Machine/software information

[1] demananalysis (wang's algorithm)
Linux 2.6.12-1.1380_FC3 #1 i386 GNU/Linux AMD
Athlon XP 1900+ CPU (1.6 GHz) 1 GB RAM

[2] demanalysis (domingue's algorithm)
Linux 2.6.12-1.1380_FC3 #1 i386 GNU/Linux AMD
Athlon XP 1900+ CPU (1.6 GHz) 1 GB RAM

[3] ARC/INFO Workstation 9.1 (drainage analysis) +
Matlab 6.5 (FFT based noise generation).
Windows 2000 SP 4, Pentium 4, 3.06 GHz, 1.5 GB RAM

[4] demanalysis (wang's algorithm)
Windows 2000 SP 4, Pentium 4, 3.06 GHz, 1.5 GB RAM

[5] demanalysis (wang's algorithm)
UltraSparc IIi Sabre (64bit) 512 MB RAM
```