Tartu Ülikool
Matemaatika-informaatikateaduskond
Matemaatilise Statistika Instituut

Valeri Koort

Võrestiku-, Monte-Carlo ning kvaasi-Monte-Carlo meetodid Euroopa optsioonide hindamiseks

Magistritöö

Juhendajad:
Raul Kangro,
Tõnu Kollo

Tartu
2004

Sisukord

Sissejuhatus

Optsioon on tuletisväärtpaber — tema hind on määratud mõnede teiste väärtpaberite (aktsiate) hindade käitumise poolt.

Üsna laialdastel eeldustel, näiteks Black–Scholesi mudeli kehtimisel, osutub optsiooni hind üheselt määratuks. Valemeid optsiooni hinna määramiseks leidub aga vaid kõige triviaalsematel juhtudel, üldjuhul tuleb kasutusele võtta erinevad arvutusmeetodid. Meetodeid on palju (simulatsioonimeetodid ehk Monte-Carlo meetodid, puumeetodid, võrgumeetodid, kvaasi-Monte-Carlo meetodid), neid saab täiendada, juurde tekib ka uusi võimalusi.

Käesolev töö on jätkuks ning edasiarenguks autori bakalaureusetööle [8] — vaatluse alla on võetud uued meetodid ning on vaadeldud ka kolmest aktsiast sõltuvaid optsioone. Töö eesmärgiks on uurida uut paljutõotavat meetodit — kvaasi-Monte-Carlo meetodit, realiseerida see programmina ja võrrelda konkreetse näite korral saavutatud koonduvuskiirust Monte-Carlo meetodi ning võrestikumeetodiga, samas mitmeti täiendades vastavaid programme.

Autori panus seisneb paljude väidete kontrollimises konkreetsel juhul, eri valdkondadest teadaolevate tulemuste ühildamises, mõnede tõestuste iseseisvas läbiviimises, programmide kirjutamises ning numbriliste tulemuste analüüsimises.

1. Optsioonide hindamisest

1.1. Põhimõisted

Antud töö tegeleb *väärtpaberi optsiooni* hindamisega. Tuletame meelde selle definitsiooni.

Definitsioon 1.1. ***Väärtpaberi optsioon*** *on kokkulepe, mis annab omanikule õiguse osta (ostuoptsioon) või müüa (müügioptsioon) väärtpaberit kindlaksmääratud hinnaga kindlaksmääratud ajaperioodi jooksul. Mitme väärtpaberi optsioon annab vastavad õigused mitme väärtpaberi jaoks.*

Käesolev töö piirdub Euroopa tüüpi optsioonidega, st. nendega, mis võimaldavad realiseerida ostu- või müügiõigust ainult ühel hetkel — täitmisajal.

Optsioonide põhjalikum tutvustus kood näidetega on esitatud bakalaureusetöös [8], samuti viitame sealsele kirjanduse loetelule. Põhilised optsioonidega ja finantsturuga seotud mõisted on järgmised.

Alusväärtpaberid on väärtpaberid, mille kohta on optsioonileping koostatud. Selles töös vaatleme vaid selliseid väärtpabereid, mis ei kanna dividende. Tähistagu $S_i(t)$ i–nda alusväärtpaberi hinda ajal t.

Täitmisaeg (realiseerimisaeg) T on optsiooni kehtimise lõppaeg.

Täitmishinnad (realiseerimishinnad) K_i on hinnad, millega võib lepingu kohaselt alusväärtpabereid realiseerimismomendil osta või müüa.

Maksefunktsioon p on funktsioon, mille väärtus on määratud aktsiahinna ajalooga optsiooni realiseerimismomendini ning mis näitab tulu suurust, mida võib optsiooni realiseerimisel saada.

Volatiilsus (varieeruvus, variatsioon) σ on arvuline näitaja, mis kirjeldab väärtpaberi hinna stabiilsust. Mida väiksem on volatiilsus (mis on alt tõkestatud nulliga), seda vähem kõigub väärtpaberi hind. Tüüpiline volatiilsuse väärtus asub 0.20 ja 0.40 vahel.

Pangaprotsent (riskivaba intressimäär) r on raha juurdekasv, mis saadakse rahasumma hoiustamisel pangas (esitatud protsentides esialgse summa suhtes). Antud juhul vaadeldav lühiajalise hoiuse või laenu protsendina. Tüüpiline väärtus aasta kohta on ligikaudu 0.08 ehk 8%.

Väärtpaberi kasumimäär μ on keskmine oodatav raha juurdekasv, mida võib saada rahasumma investeerimisel väärtpaberisse. Tavaliselt on see aasta kohta ligikaudu 0.16 ehk 16%. Kasumimäära, mis vastab i–ndale väärtpaberile, tähistame μ_i .

Wieneri protsess $z(t)$ (Browni liikumine) on juhuslik protsess $\big(z(t)\big)_{0\leqslant t\leqslant\infty}$, mille muut $\Delta z(t)$ ajavahemiku Δt jooksul rahuldab kahte tingimust:

1. $\Delta z(t) \sim \mathrm{N}(0, \sqrt{\Delta t})$;
2. $\Delta z(t)$ väärtused kahe erineva (mittelõikuva) intervalli Δt jaoks on sõltumatud.

1.2. Black–Scholesi teooria

Black–Scholesi aktsiaturu mudelist läheb vaja eeskätt võrrandeid väärtpaberite hindade jaoks. Enamasti eeldatakse, et need hinnad rahuldavad stohhastilisi diferentsiaalvõrrandeid

$$\mathrm{d}S_i = S_i\Big(\mu_i \mathrm{d}t + \sum_{j=1}^{n} \sigma_{ij} \mathrm{d}z_j\Big), \qquad (i = 1, \ldots, n) \tag{1.1}$$

kus $\mathrm{d}S_i$ on i–nda väärtpaberi hinna muut ajamuudu $\mathrm{d}t$ jooksul, μ_i on i–nda väärtpaberi hinna keskmine protsentuaalne juurdekasv, S_i on i–nda väärtpaberi hind, σ_{ij} on i–ndale väärtpaberile vastavad volatiilsused ning $\mathrm{d}z_j$ on j–nda Wieneri protsessi muut ($z_j\,,\ j = 1, \ldots, n$ — sõltumatud Wieneri protsessid).

Seosed (1.1) saab kirja panna ka alternatiivsel kujul:

$$\mathrm{d}S_i = S_i\big(\mu_i \mathrm{d}t + \sigma_i\, \mathrm{d}B_i(t)\big)\,, \qquad (i = 1, \ldots, n), \tag{1.2}$$

kus $\mathrm{d}B_i(t)$ on sõltuvate Wieneri protsesside muudud. Vastavad korrelatsioonikordajad ning suurused σ_i on väljaarvutatavad suurustest σ_{ij} ; juhul $n = 2$ on see tehtud töös [8]. Samas töös on näidatud, et optsiooni hindamise seisukohalt saab võtta $\mu_i = r$ iga i korral. Seega seosed (1.2) omandavad kuju

$$\mathrm{d}S_i = S_i\big(r\mathrm{d}t + \sigma_i\, \mathrm{d}B_i(t)\big)\,, \qquad (i = 1, \ldots, n). \tag{1.3}$$

Itô valemi abil (lähemalt vt. [8]) saame leida logaritmitud hindade muute:

$$\mathrm{d}\ln S_i = \left(r - \frac{\sigma_i^2}{2}\right)\mathrm{d}t + \sigma_i\, \mathrm{d}B_i(t)\,, \qquad (i = 1, \ldots, n)$$

ning edasi, integreerides neid avaldisi üle lõigu $[\,0, T\,]$, saame

$$\ln \frac{S_i(T)}{S_i(0)} = \left(r - \frac{\sigma_i^2}{2}\right) T + \sigma_i\, B_i(T)\,, \qquad (i = 1, \ldots, n). \tag{1.4}$$

Siit saame seosed, mida edaspidi kasutame väärtpaberite hindade modelleerimiseks:

$$S_i(T) = S_i(0) \exp\left(\big(r - \tfrac{\sigma_i^2}{2}\big)T + \sigma_i\, B_i(T)\right), \qquad (i = 1, \ldots, n). \tag{1.5}$$

Wieneri protsesside vektor on nüüd mitmemõõtmelise normaaljaotusega:

$$\boldsymbol{B}(T) = \begin{pmatrix} B_1(T) \\ \ldots \\ B_n(T) \end{pmatrix} \sim \mathrm{N}_n(\mathbf{0}, \boldsymbol{\Sigma})\,, \qquad \boldsymbol{\Sigma} = T \begin{pmatrix} 1 & \rho_{12} & \cdots & \rho_{1n} \\ \rho_{12} & 1 & \cdots & \rho_{2_n} \\ \cdots & \cdots & \cdots & \cdots \\ \rho_{1n} & \cdots & \rho_{n-1\,n} & 1 \end{pmatrix}, \tag{1.6}$$

kus korrelatsioonikordajad on kujul

$$\rho\Big(\sum_{k=1}^{n} \sigma_{ik} \mathrm{d}z_k(t), \sum_{k=1}^{n} \sigma_{jk} \mathrm{d}z_k(t)\Big) = \rho_{ij} = \rho\big(\sigma_i \mathrm{d}B_i(t), \sigma_j \mathrm{d}B_j(t)\big) = \rho\big(\sigma_i B_i(T), \sigma_j B_j(T)\big). \tag{1.7}$$

Võrrandite (1.1) ja (1.2) ekvivalentsusest järelduvad seosed kordajate σ_i leidmiseks:

$$\mathrm{D}\Big(\sum_{k=1}^{n} \sigma_{ik} \mathrm{d}z_k(t)\Big) = \mathrm{D}\big(\sigma_i \mathrm{d}B_i(t)\big). \tag{1.8}$$

Kahemõõtmelisel juhul selgub, et

$$\rho = \frac{\sigma_{11}\sigma_{21} + \sigma_{12}\sigma_{22}}{\sqrt{(\sigma_{11}^2 + \sigma_{12}^2)(\sigma_{21}^2 + \sigma_{22}^2)}}\,, \qquad \sigma_i = \sqrt{\sigma_{i1}^2 + \sigma_{i2}^2}\,, \qquad i = 1, 2.$$

Üldjuhul annavad võrrandid (1.7) ja (1.8) järgmised seosed korrelatsioonide ning dispersioonikordajate jaoks:

$$\rho_{ij} = \frac{\sum_{k=1}^{n} \sigma_{ik}\sigma_{jk}}{\sigma_i \sigma_j}\,, \qquad \sigma_i = \sqrt{\sum_{k=1}^{n} \sigma_{ik}^2}\,, \qquad i, j = 1, \ldots, n.$$

1.3. Integreerimine üle ühikruudu

Järgnevalt näitame, et optsiooni hinda saab esitada integraalina üle n-mõõtmelise ühikkuupi, kus n on alusväärtpaberite arv. Kahemõõtmelisel juhul $n = 2$ korral on tegemist ühikruuduga.

Töös [8] on näidatud, et optsiooni hind on esitatav keskväärtusena:

$$\mathrm{E}\Big(e^{-rT} p\big(S_1(T), \ldots, S_n(T)\big)\Big)\,, \tag{1.9}$$

kus r on pangaprotsent, T on täitmisaeg, p on huvipakkuva optsiooni maksefunktsioon ning $S_i(T)$ on i-nda väärtpaberi hind lõppajal T. Juhuslikud on selles seoses suurused $S_i(T)$; seejuures neid väärtpaberite hindu saab genereerida seose (1.5) abil.

Üks võimalus antud keskväärtuse hindamiseks põhineb tugeva suurte arvude seaduse kasutamisel, mille kohaselt keskväärtus on lähendatav juhusliku valimi aritmeetilise keskmisega. Selleks tuleb esmalt modelleerida väärtpaberite lõpphindade sõltumatud komplektid $\big(S_1(T), \ldots, S_n(T)\big)$. See lähenemine ongi Monte-Carlo meetodi aluseks; teised selles töös vaadeldavad meetodid on aga otseselt rakendatavad integraalide leidmiseks.

Nüüd teisendame keskväärtuse avaldise integraaliks, kusjuures integreerimispiirkonnaks võtame n-mõõtmelise ühikkuupi. Kombineerides avaldisi (1.9) ning (1.5), saame kirjutada

$$\begin{aligned}
&\mathrm{E}\left(e^{-rT} p\left(S_1(0) e^{\left(r - \frac{\sigma_1^2}{2}\right)T + \sigma_1 B_1(T)}, \ldots, S_n(0) e^{\left(r - \frac{\sigma_n^2}{2}\right)T + \sigma_n B_n(T)}\right)\right) = \\
&\qquad = \mathrm{E}\left(e^{-rT} p\left(S_1(0) e^{\left(r - \frac{\sigma_1^2}{2}\right)T + \sigma_1 G_1(\boldsymbol{u})}, \ldots, S_n(0) e^{\left(r - \frac{\sigma_n^2}{2}\right)T + \sigma_n G_n(\boldsymbol{u})}\right)\right) = \\
&= \int_0^1 \cdots \int_0^1 e^{-rT} p\left(S_1(0) e^{\left(r - \frac{\sigma_1^2}{2}\right)T + \sigma_1 G_1(\boldsymbol{u})}, \ldots, S_n(0) e^{\left(r - \frac{\sigma_n^2}{2}\right)T + \sigma_n G_n(\boldsymbol{u})}\right) f_{\boldsymbol{U}}(\boldsymbol{u}) \mathrm{d}u_1 \cdots \mathrm{d}u_n = \\
&\qquad = \int_{[0,1]^n} e^{-rT} p\left(S_1(0) e^{\left(r - \frac{\sigma_1^2}{2}\right)T + \sigma_1 G_1(\boldsymbol{u})}, \ldots, S_n(0) e^{\left(r - \frac{\sigma_n^2}{2}\right)T + \sigma_n G_n(\boldsymbol{u})}\right) f_{\boldsymbol{U}}(\boldsymbol{u}) \mathrm{d}\boldsymbol{u}\,,
\end{aligned} \tag{1.10}$$

kus $\boldsymbol{U} = (U_1, \ldots, U_n)^{\mathrm{T}}$ on n-mõõtmelise ühtlase jaotusega juhuslik suurus tihedusfunktsiooniga $f_{\boldsymbol{U}}$, $\boldsymbol{u} = (u_1, \ldots, u_n)^{\mathrm{T}}$ on punkt n-mõõtmelisest ühikkuubist ning n argumendi funktsioon $G(\boldsymbol{u}) = \big(G_1(\boldsymbol{u}), \ldots, G_n(\boldsymbol{u})\big)^{\mathrm{T}}$ on selline, et kui vektor $\boldsymbol{U}$ on n-mõõtmelise ühtlase jaotusega $\mathrm{U}_n(0,1)$, siis vektor $G(\boldsymbol{U})$ on n-mõõtmelise normaaljaotusega $\mathrm{N}_n(\boldsymbol{0}, \boldsymbol{\Sigma})$ — st. Wieneri protsesside vektori $\big(B_1(T), \ldots, B_n(T)\big)^{\mathrm{T}}$ jaotusega valemist (1.6). Sellise funktsiooni konstrueerimine on kirjeldatud osas 4.4.

Saadud integraali (1.10) saab hinnata pseudojuhusliku valimi keskmisega, kuna

$$\frac{1}{n}\sum_{i=1}^{n} g(u_i) \to \mathrm{E}g(U) = \int_0^1 g(x) f_U(x)\mathrm{d}x\,,$$

kus u_i on pseudojuhuslikud arvud jaotusest tihedusega f_U, selleks jaotuseks on antud juhul $\mathrm{U}(0,1)$.

Optsiooni hind on seega esitatav keskväärtusena integraali abil. Monte-Carlo meetod käsitleb seda hinda enamasti keskväärtusena, võrestiku- ja kvaasi-Monte-Carlo meetodid aga integraalina. Ometi selgub, et kõik kolm meetodit töötavad praktiliselt sama skeemi järgi: ühikruudust võetakse punktid, arvutatakse funktsiooni väärtused nendes punktides ja saadud arvude keskmine ongi lähendiks. Erinevus on ainult selles, kuidas genereeritakse esialgsed punktid. Siiski leiab see sarnasus aset tänu vaadeldava võrestikumeetodi lihtsusele; keerulisemate ja täpsemate lähenemiste korral muutub võrestikumeetodi algoritm komplitseeritumaks.

1.4. Monte-Carlo meetodi veahinnangust

Monte-Carlo meetod lähendab optsiooni hinda pseudojuhusliku valimi keskväärtusega, koondumise taga on tugev suurte arvude seadus. Kuna valim on väga suure mahuga ning koosneb sõltumatutest sama jaotusega juhuslikest suurustest, siis valimi keskmine on tsentraalse piirteoreemi tõttu ligikaudu normaaljaotusega. Selle normaaljaotuse parameetreid saab hinnata ning konstrueerida valimi keskmisele usaldusvahemiku. Usaldusvahemiku abil saab hinnata tõenäosust, et lähendi viga ei ületa absoluutväärtuselt teatud etteantud arvu.

Seega Monte-Carlo meetodi veahinnang on tõenäosuslik: saab väita, et viga ei ületa etteantud arvu teatud tõenäosusega. Antud teemat on põhjalikult käsitletud töös [8].

1.5. Antiteetiliste arvude kasutamine

Antiteetiliste juhuslike arvude kasutamine on üks võimalusi Monte-Carlo meetodi kiirendamiseks. Pikem kirjeldus ja näited ühemõõtmelisel juhul on esitatud bakalaureusetöös [8].

Meetodi idee seisneb selles, et olles genereerinud ühe pseudojuhusliku suuruse väärtuse z jaotusest $\mathrm{N}(0,1)$, võime kohe võtta veel ühe väärtuse, nimelt $-z$, kuna ka see on vaadeldav sama jaotuse väärtusena. Mitmemõõtmelisel juhul on võimalusi rohkem: paarist (z_1, z_2) saab neli paari antiteetilise meetodi jaoks — (z_1, z_2), $(z_1, -z_2)$, $(-z_1, z_2)$ ning $(-z_1, -z_2)$; kolmemõõtmelisel juhul saab ühest kolmikust 8 uut ning üldjuhul n-mõõtmelisel juhul ühe vektori asemel 2^n uut vektorit.

Lisaks märgi muutmisele võib põhimõtteliselt kasutada ka vektori koordinaatide ümberjärjestamist — näiteks kahemõõtmelisel juhul lisanduvad siis veel vektorid (z_2, z_1), $(z_2, -z_1)$, $(-z_2, z_1)$ ja $(-z_2, -z_1)$. Üldjuhul on ümberjärjestusi kokku $n!$ tükki; seega ühest vektorist saame kokkuvõttes $2^n n!$ vektorit. Aga praktika näitab, et ümberjärjestused ei vähenda rohkem dispersiooni, küll aga nõuavad lisaaega; seetõttu on käesolevas töös kasutatud ainult märgimuudatusi, millega on saadud ühest vektorist $2^n - 1$ antiteetilist vektorit juurde.

2. Kvaasi-Monte-Carlo meetod

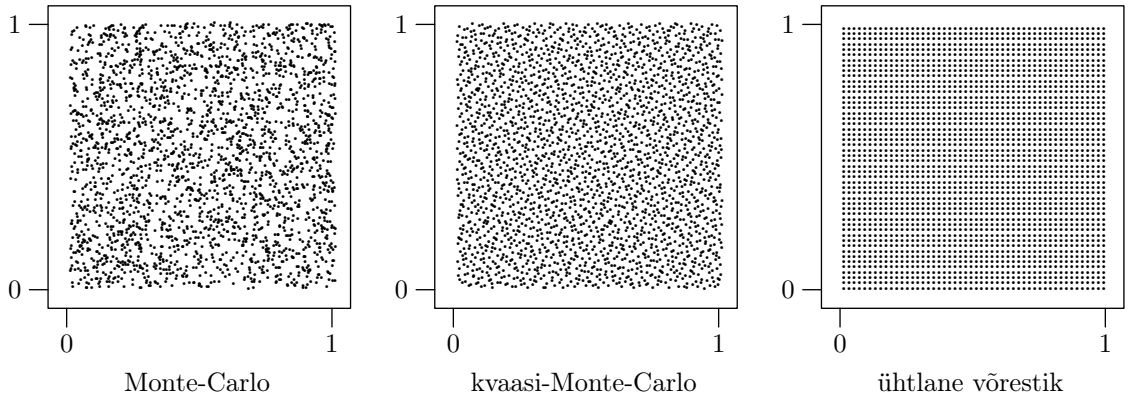
2.1. Sissejuhatus

Meie ülesandeks on jätkuvalt integreerimine üle ühikruudu. Selleks tuleb meil arvutada huvipakkuva funktsiooni väärtused mõnedes valitud punktides. Kuidas neid punkte valida? Monte-Carlo meetod lahendab seda probleemi juhuslikkuse abil: punktid valitakse juhuslike arvude abil (vähemalt teoreetiliselt on need punktid juhuslikud; praktikas on nad reeglina siiski pseudojuhuslikud).

Kvaasi-Monte-Carlo meetod ei sisalda üldse juhuslikkust. Punktid valitakse üksteise järel determineeritud algoritmi kohaselt. Ja kuigi saadavad arvud läbivad mõnda juhuslikkuse testi rahuldavalt (nt. empiirilise jaotusega on kõik korras), ei rahulda nad seeriate testi (juhuslikkuse testidest on lähemalt juttu õpikus [7]).

Mille poolest see meetod hea on?

Esiteks, algoritmi omaduste tõttu on punktide paiknemine väga ühtlane, võrreldav ühtlase võrestiku punktidega. See on selge eelis Monte-Carlo meetodi ees, kus punktid kipuvad ühes piirkonnas moodustama tihedamaid kooslusi, mõni teine piirkond jääb aga hõredaks. Seda olukorda illustreerib joonis 2.1.



Joonis 2.1. Eri meeetodite abil genereeritud 2 500 punkti ühikruudus.

Teiseks säilitab kvaasi-Monte-Carlo meetod ühe hea Monte-Carlo meetodi omaduse: integraali arvutamist võib lõpetada ükskõik millal — tulemus on täiesti mõistlik. Selline omadus ei säili võrestiku punktide kasutamisel, kuna näiteks poole võrestiku peal katkestamine põhjustaks suurt kõrvalekallet arvutatavas integraali väärtuses.

Seega kvaasi-Monte-Carlo meetod on võtnud ühe hea omaduse Monte-Carlo meetodilt (kiire lõpetamise võimalus) ning teise hea omaduse võrestikumeetodilt (punktide suur ühtlasus).

Kvaasi-Monte-Carlo meetodis kasutatavaid punkte nimetatakse *madala hälbimisega jadaks* (ingl. *low discrepancy sequence*). Anname sellise jada täpse definitsiooni, eelnevalt defineerides hälbimise.

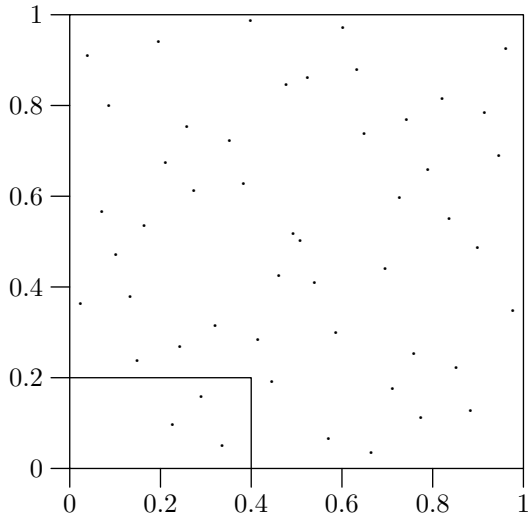
Definitsioon 2.1. *Olgu X_n punktide x_i hulk, $X_n = \{x_1, \ldots, x_n\}$, punktid x_i olgu d-mõõtmelisest kuubist $[\,0,1\,]^d$ ning olgu J hulga $[\,0,1\,]^d$ alamhulk mõõduga $V(J)$. Tähistagu $A(J)$ hulga X_n selliste punktide arvu, mis on hulga J sees. Siis hulga X_n hälbimiseks (ka täht-hälbimiseks) nimetatakse suurust*

$$D^* = D^*(X_n) = D^*(x_1, \ldots, x_n) = \sup_J \left|\tfrac{1}{n}A(J) - V(J)\right|, \tag{2.1}$$

kus supreemum võetakse üle kõigi selliste piirkondade J, mis on kujul $\prod_{i=1}^{d}[0, y_i)$, $0 < y_i \leqslant 1$.

Lõpmatu jada X korral kasutame tema n esimese punkti hälbimise tähisena $D^*(X, n)$.

Hälbimine on seega punktide ühtlasuse arvuline näitaja. Illustreerime definitsiooni 2.1 kahemõõtmelise juhu jaoks joonisega 2.2.



Joonis 2.2. Näide hälbimise definitsiooni juurde.

Ühikruudus on $n = 50$ punkti. Hulgaks J on $[0, 0.4) \times [0, 0.2)$. Siis $V(J) = 0.08$ ning $A(J)/n = 3/50 = 0.06$. Hälbimise arvutamiseks tuleks võtta supreemum (2.1) üle kõigi nullis algavate ristkülikute.

Hälbimine on d-mõõtmeline üldistus Kolmogorov–Smirnovi kaugusele.

Ühtlase jaotuse $\mathrm{U}[0, 1)^d$ kohta on näidatud, et

$$\limsup_{n\to\infty} \frac{\sqrt{2n}D^*(X, n)}{\sqrt{\ln\ln n}} = 1 \quad \text{p. k.},$$

kust tõenäosusega 1

$$D^*(X) = \mathrm{O}\left(\sqrt{\frac{\ln\ln n}{n}}\right),$$

kus $x_i \sim \mathrm{U}[0, 1)^d$ on omavahel sõltumatud. Kuna kahekordne logaritm kasvab väga aeglaselt, on hälbimine ligikaudu suurusjärku $n^{-1/2}$ suurte n korral.

Kuid punktide x_i oskusliku valikuga saab hälbimise teha palju väiksemaks. On olemas lõpmatud jadad, mille korral

$$D^* = \mathrm{O}\left(\frac{(\ln n)^d}{n}\right). \tag{2.2}$$

Arvatakse, et ei ole võimalik konstrueerida lõpmatut jada hälbimisega $D^* = \mathrm{o}\big((\ln n)^d/n\big)$. Viimane hüpotees on tõestatud juhtude $d = 1; 2$ korral.

Definitsioon 2.2. *Madala hälbimisega jadaks nimetatakse lõpmatut jada X, mille korral on täidetud tingimus*

$$D^*(X, n) \leqslant C(X, d)\frac{(\ln n)^d}{n}, \qquad (n \geqslant 2) \tag{2.3}$$

kus konstant C sõltub jadast X ning mõõtmete arvust d.

Järgnevalt kirjeldame kaht punktide genereerimise algoritmi, mida kasutatakse Lisades A ja B toodud programmides; valdav osa esitusest põhineb allikatel [4] ja [11].

2.2. Haltoni jada

Haltoni algoritm (sisse toodud töös [5]) on üks lihtsamaid näiteid madala hälbimisega jadast.

Esiteks läheb vaja d erinevat algarvu $p_1, \ldots, p_n$; reeglina võetakse lihtsuse mõttes d esimest algarvu. Iga naturaalarv k on üheselt esitatav arvu p_i astmete lineaarkombinatsioonina:

$$k = \sum_{j=1}^{\lceil \log_{p_i} k \rceil} a_j p_i^j \,,$$

kus a_j on täisarv, $0 \leqslant a_j \leqslant p_j - 1$.

Ruut-pöördfunktsioon ϕ_{p_i} defineeritakse võrdusega

$$\phi_{p_i}(k) = \sum_{j=1}^{\lceil \log_{p_i} k \rceil} a_j p_i^{-j-1}$$

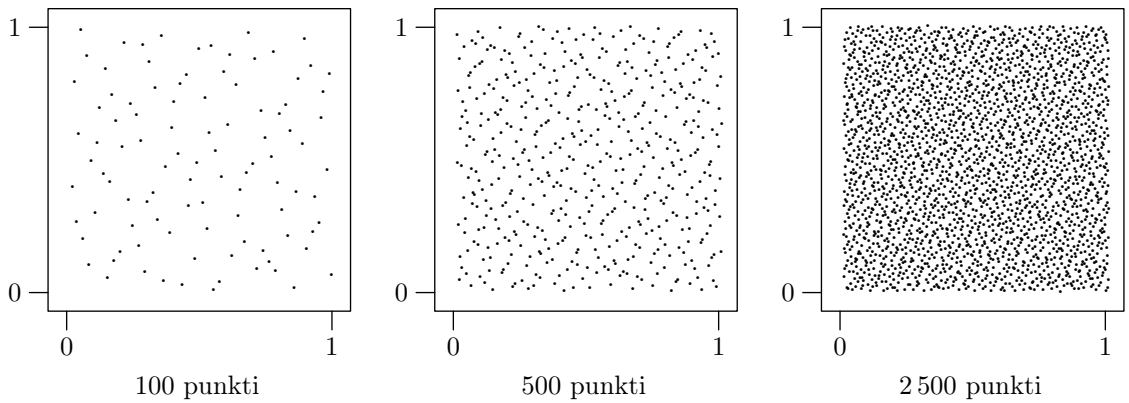
ning d-mõõtmelise Haltoni jada punktid x_i saadakse järgmiselt:

$$x_i = \big(\phi_{p_1}(k), \phi_{p_2}(k), \ldots, \phi_{p_d}(k)\big), \qquad i \geqslant 0.$$

Sisuliselt realiseerivad need võrdused järgmise algoritmi:

1. arv k kirjutatakse positsioonilises arvusüsteemis alusega p_i (näiteks $k = 17$ alusel $p_2 = 3$ on 122);
2. kirjutatakse saadud arvu numbrid tagant ettepoole ning pannakse ette kümnendkoma (näites saame 0.221);
3. saadud arv on k-nda arvu x_k i-s koordinaat positsioonilises arvusüsteemis alusega p_i, jääb üle vaid tuua ta tagasi kümnendsüsteemi (0.221_3 on $0.(925)_{10}$).

Esimeste Haltoni algoritmi abil genereeritud punktide paiknemist illustreerib joonis 2.3.



Joonis 2.3. Haltoni jada 100, 500 ja 2 500 esimest punkti juhul $d = 2$.

2.3. Soboli jada

Soboli jada genereerimise algoritm on tutvustatud töös [12] ning seisneb lühidalt järgnevas.

Esiteks defineerime ühe vajamineva lisamõiste.

Definitsioon 2.3. *Primitiivseks polünoomiks mooduli* 2 *järgi nimetatakse ühemuutuja polünoomi, mille kordajad on hulgast* $\{0,1\}$ *ning mis genereerib pikima võimaliku rekurrentse bittide jada; nimelt, kui polünoomiks on*

$$x^n + a_1 x^{n-1} + ... + a_{n-1}x + 1\,,$$

siis tsükliline rekurrentne jada $\{x_i\}$*, kus*

$$x_k = x_{k-n} \oplus a_1 x_{k-n+1} \oplus \cdots \oplus a_{n-2}x_{k-2} \oplus a_{n-1}x_{k-1}\,,$$

on maksimaalse võimaliku pikkusega, st. pikkusega $2^n - 1$ *(*$\oplus$ *tähistab bitikaupa* XOR*-operaatorit* [1]*; bittide initsialiseerimiskomplekt* $x_1, ..., x_n$ *võib olla suvaline, välja arvatud kõik nullid).*

Järgnevalt on toodud mõningad näited seda tüüpi polünoomidest:

$$x+1, \quad x^2+x+1, \quad x^3+x+1, \quad x^4+x+1, \quad x^4+x^3+1, \quad x^5+x^2+1,$$
$$x^8+x^4+x^3+x^2+1, \quad x^{100}+x^8+x^7+x^2+1.$$

(Primitiivsetest polünoomidest mooduli 2 järgi on lähemalt juttu raamatutes [4] ning [13].)

Vaatleme alguses ühemõõtmelist juhtu, $d = 1$.

Olgu $v_1, ..., v_w$ binaarsed murrud w bitiga pärast koma. Suurusi v_i nimetatakse *suunanumbriteks*; nende genereerimist vaatleme pisut hiljem.

Soboli originaalalgoritmis genereeritakse ühemõõtmeline jada kui

$$x_k = b_1 v_1 \oplus b_2 v_2 \oplus \cdots \oplus b_{w^*} v_{w^*}\,, \qquad k \geqslant 0,$$

kus arvud b_j on võetud arvu k binaarsest esitusest $k = \sum_{i=0}^{\lceil \log_2 k \rceil} b_{i+1} 2^i$ ning suurimaks indeksiks on $w^* = \min(w, \lceil \log_2 k \rceil + 1)$. Antonov ja Saleev [2] näitasid, et arvu k bittide asemel saab edukalt kasutada arvu k Gray koodi $G(k)$ bitte [2]. Selline muudatus suurendab oluliselt algoritmi töökiirust.

Ja nüüd suunanumbrite genereerimisest. Arvud v_i saadakse kasutades primitiivseid polünoome mooduli 2 järgi. Olgu selliseks polünoomiks

$$P(x) = x^n + a_1 x^{n-1} + ... + a_{n-1}x + 1\,.$$

Siis esmalt genereeritakse täisarvud m_i järgmise rekurrentse seose abil

$$m_j = 2a_1 m_{j-1} \oplus 2^2 a_2 m_{j-2} \oplus \cdots \oplus 2^{n-1} a_{n-1} m_{j-n+1} \oplus (2^n m_{j-n} \oplus m_{j-n}), \qquad j > n;$$

[1] Kahe biti XOR-operaatori e. bitikaupa välistava OR-operaatori väärtus on defineeritud järgnevalt:

$$0 \oplus 0 = 0, \quad 0 \oplus 1 = 1, \quad 1 \oplus 0 = 1, \quad 1 \oplus 1 = 0.$$

[2] Gray koodi tähtsaimaks omaduseks on see, et $G(k)$ ja $G(k+1)$ binaarsed esitused erinevad täpselt ühe biti võrra (lähemalt vt. [4]). Arvu k Gray koodi genereerimine on väga lihtne:

$$G(k) = k \oplus \lfloor k/2 \rfloor.$$

Näiteks $G(11) = 11 \oplus \lfloor 11/2 \rfloor = 11 \oplus 5 = 1011_2 \oplus 0101_2 = 1110_2$.

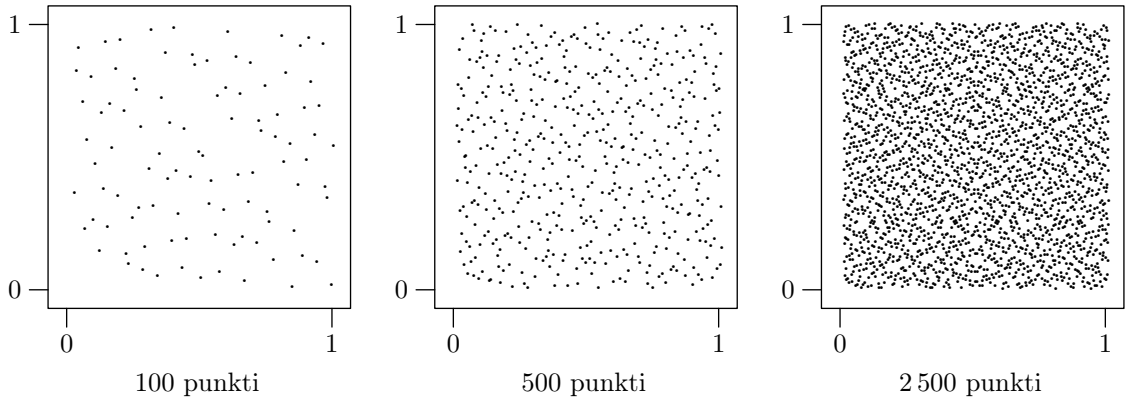
algväärtusteks $m_1, \ldots, m_n$ võivad olla suvalised paaritud täisarvud, mis on tõkestatud seosega $0 < m_i < 2^i$. Suunanumbrid ise on arvutatavad seosest

$$v_i = \frac{m_i}{2^i}\,, \qquad i = 1, \ldots, w\,.$$

Nüüd saab konstrueerida Soboli jada suvalise dimensiooni d korral. Nimelt, olgu polünoomid $P_1, \ldots, P_d$ erinevad primitiivsed polünoomid mooduli 2 järgi; polünoomi P_i poolt genereeritavat ühemõõtmelist Soboli jada tähistame $\{x_k^i\}_{k=1}^{\infty}$. Siis d-mõõtmeliste Soboli punktide jada on defineeritud seosega

$$x_k = (x_k^1, x_k^2, \ldots, x_k^d).$$

Esimeste Soboli algoritmi abil genereeritud punktide paiknemist illustreerib joonis 2.4.



Joonis 2.4. Soboli jada 100, 500 ja 2 500 esimest punkti juhul $d = 2$.

2.4. Vea hindamisest

Mäletatavasti soovime hinnata integraali I väärtust ning kasutame selleks aritmeetilist keskmist $\hat{I}$:

$$I = \int\limits_{[0,1]^d} f(x)\mathrm{d}x\,, \qquad \hat{I} = \frac{1}{n}\sum_{i=1}^{n} f(x_i)\,,$$

kus x_i on mõne madala hälbimisega jada punktid.

Parim teadaolev seos vea hindamiseks on Koksma–Hlawka võrratus

$$|\hat{I} - I| \leqslant D^*(x_1, \ldots, x_n) V_{\mathrm{HK}}(f), \tag{2.4}$$

kus V_{HK} on funktsiooni f koguvarieeruvus Hardy ja Krause mõttes (täpne definitsioon on raamatus [9]).

Kokkuvõttes järeldub seostest (2.3) ning (2.4), et kvaasi-Monte-Carlo meetod võib olla tunduvalt parem Monte-Carlo meetodist (mille tehtav viga oli mäletatavasti suurusjärku $n^{-1/2}$), juhul kui n on piisavalt suur ning huvipakkuv funktsioon f on lõpliku koguvarieeruvusega V_{HK}. Nii enamasti juhtubki, kuid võrratus (2.4) ei ole kuigi hea praktiliseks kasutamiseks. Raskused on suuruse D^* arvutamisega, varieeruvusega V_{HK} on asi veelgi halvem. On juhte, kus V_{HK} on lõpmatu, aga kvaasi-Monte-Carlo meetod on ikkagi Monte-Carlo omast kiirem.

Seega teoreetilistest tulemustest pole antud juhul suurt abi; õnneks kvaasi-Monte-Carlo meetodite koondumine on Monte-Carlo meetodiga võrreldes palju stabiilsem, nii

et saab suhteliselt edukalt kasutada mõnda lihtsat kriteeriumi ja lõpetada programmi töö «kasutaja eksperthinnangu põhjal». Näiteks käesoleva töö raames vaadeldud näidete korral (arvulised tulemused on punktis 5) osutus rahuldavaks järgmine kriteerium: piisav täpsus ε on saavutatud n punkti korral, kui on rahuldatud võrratus

$$|I_n - I_{n-10\,000}| < 0.1\varepsilon\,,$$

kus I_k on integraali lähend k punkti korral.

2.5. Veel võimalusi

Peale Haltoni ja Soboli jada leidub veel mitu võimalust madala hälbimisega jadade saamiseks. Näiteks üldistatud Faure jada [10], Niederreiteri ja Xingi väljatöötatud jadad. Väga põhjalik ülevaade nende võimaluste kohta on antud artiklis [3].

Meetodite võrdlemise osas tuleks kindlasti esile tuua artiklit [11], mis võrdleb Monte-Carlo ning kvaasi-Monte-Carlo meetodeid eriti kõrge dimensiooniga integraalide arvutamisel (kuni 360). Vaatluse alla oli võetud Monte-Carlo, kvaasi-Monte-Carlo (Haltoni ja Soboli algoritmid) ning antiteetilistega suurustega Monte-Carlo meetodid. Kokkuvõttes osutus, et:

- Soboli algoritm on selgelt üle Monte-Carlo meetodist, ta koondub kiiremini ja stabiilsemalt.
- Soboli algoritm on selgelt üle Haltoni algoritmist.
- Monte-Carlo meetod on tundlik initsialiseerimise suhtes.
- Soboli algoritm on selgelt üle ka antiteetiliste suurustega Monte-Carlo meetodist, mis on omakorda selgelt üle Monte-Carlo meetodist.
- Kvaasi-Monte-Carlo meetodite lõpetamiskriteeriumina kasutatakse lihtsat reeglit: «kui saadud väärtus enam eriti palju ei muutu, siis lõpetame ära».

Muuhulgas reklaamib see artikkel autorite kirjutatud tarkvara integraalide arvutamiseks uuritud meetodite abil.

Paljude teiste võrdluste tulemuseks on konstateerimine, et mõnikord on kvaasi-Monte-Carlo meetod Monte-Carlo meetodist tunduvalt kiirem, mõnikord töötavad meetodid võrreldava kiirusega.

Mitmed Monte-Carlo meetodite jaoks tuntud kiirendamise ja dispersiooni vähendamise võimalused kehtivad osaliselt ka kvaasi-Monte-Carlo meetodi korral. Näiteks kontrollmuutujate kasutamine on võimalik ka kvaasi-Monte-Carlo meetodi korral, kuid lähenemise detailid tuleks muuta, muidu kontrollmuutujad toovad hoopis kahju (lähemalt vt. [6]).

Et kombineerida kvaasi-Monte-Carlo meetodi täpsuse hindamist Monte-Carlo meetodi praktiliste veahindamismeetoditega, töötati välja randomiseeritud kvaasi-Monte-Carlo meetod. Veahinnang selle meetodi korral on praktiliselt kasutatav, kuid tõenäosuslik.

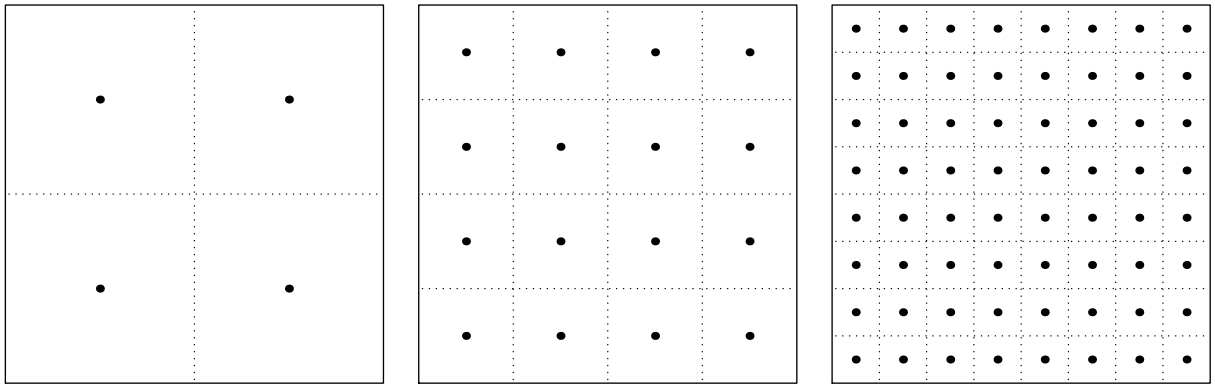
3. Võrestikumeetod

Võrestikumeetod oleks antud juhul teine äärmus: Monte-Carlo meetod põhineb juhuslikkusel (mis praktikas tähendab küll pseudojuhuslikkust), kvaasi-Monte-Carlo on otseses tähenduses pseudojuhuslik, ta oleks nagu juhuslikkuse ja determineerituse vahepeal. Determineerituse eredaks esindajaks on võrestikumeetod, kus juhuslikkus puudub üldse. Selge see, et neid teatud mõttes järjestatavaid meetodeid oleks huvitav omavahel võrrelda.

Nagu nimetusestki nähtub, ei võeta punkte juhuslikult, vaid teatud (reeglina üsna lihtsast) võrestikust lähtudes. Võrestiku valikuks on palju võimalusi; siinkohal vaatleme paari lihtsamat võimalust. Ülevaatlikkuse huvides on nad toodud kahemõõtmelise juhu jaoks; üldistamine mitmemõõtmelisele juhule on triviaalne.

3.1. Esimene meetod

Järgmine lähenemine on lihtsaim ning «klassikaline». Alustatakse sellest, et ruudu küljed jagatakse kaheks. Igal järgmisel sammul tihendatakse võrestikku kaks korda, eelmise sammu punktid visatakse ära; kolm esimest sammu on esitatud joonisel 3.1.



Joonis 3.1. Esimese võrestikumeetodi kolm esimest sammu.

Arvulised parameetrid, mis iseloomustavad antud meetodit, on koondatud tabelisse 3.1; neid saab leida elementaarse induktsiooniga.

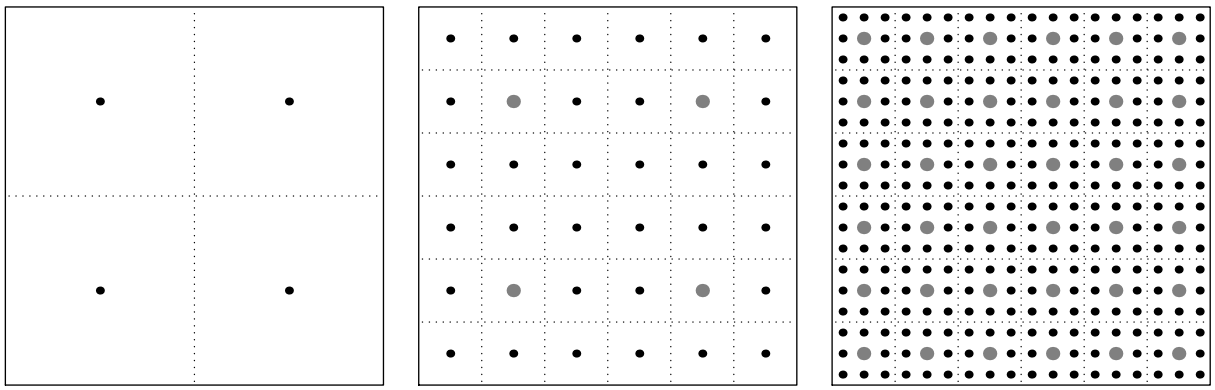
samm	1	2	3	k	s	$s+1$
punktide arv servas	2	4	8	2^k	m_s	$2m_s$
punktide koguarv	4	16	64	2^{2k}	m_s^2	$(2m_s)^2$

Tabel 3.1. Esimese võrestikumeetodi arvulised näitajad.

Võrestiku sõlmpunktide koordinaatide arvutamine on lihtne: iga üksik koordinaat omab väärtusi arvust $\frac{1}{2m_s}$ arvuni $1-\frac{1}{2m_s}$ sammuga $\frac{1}{m_s}$, kus m_s on punktide arv võrestiku servas s-ndal sammul.

3.2. Teine meetod

Esmalt anname joonisel 3.2 ülevaate selle lähenemisviisi võrestikust.



Joonis 3.2. Teise võrestikumeetodi kolm esimest sammu.

Võrreldes eelnevaga on sel meetodil nii eeliseid kui ka puudusi. Võrestiku tihenemine on kiirem, sellest tuleneb nii täpsuse kui ka tööaja kasv. Asjaolu, et esimene meetod erinevalt teisest ei kasuta eelmise sammu tulemusi, on teise meetodi eeliseks, kuid punktide koguarvu suhteliselt kiire kasvu tõttu (mõlema meetodi korral) ei ole siiski määrav.

Arvulised parameetrid, mis iseloomustavad antud meetodit, on koondatud tabelisse 3.2.

samm	1	2	3	s	$s+1$
punktide arv servas	2	6	18	m_s	$3m_s$
punktide koguarv	4	36	324	m_s^2	$(3m_s)^2$

Tabel 3.2. Teise võrestikumeetodi arvulised näitajad.

Võrestiku sõlmpunktide koordinaatide arvutamise eeskiri, lähtudes parameetri m_s väärtusest, on sama, mis eelmise meetodi korral. Programmis ei ole mõtet uuesti arvutada funktsiooni väärtust eelmise sammu punktides (kuna nende väärtuste summa säilitatakse mälus), seega need punktid tuleks igal sammul ära jätta. Punkt kuulub eelmise sammu võrestikku, kui nii tema reanumber kui ka veerunumber annavad kolmega jagades jäägiks ühe (joonisel on need punktid hallid ning paksemad).

3.3. Veahinnang

Võrestikumeetodi puhul on mõnikord raskendatud Monte-Carlo meetodi lähenemine — töötada, kuni nõutud täpsus on saavutatud. Tihti on nii, et meetodil lastakse töötada teatud aeg, siis töö lõpetatakse ja hinnatakse vea suurust. Nimelt, iga järjekordse sammu tegemiseks vajalik aeg kasvab niivõrd kiiresti, et ühel hetkel ei jõua sammu lõpetamist enam ära oodata. (Loomulikult ei pea see paika siis, kui näiteks nõutav täpsus on väike või programmi tööaja kriteerium ei ole eriti tähtis.)

Kirjeldatud võrestikumeetod on sisuliselt mitmemõõtmeline ristkülikvalem integraalide arvutamiseks. See kvadratuurvalem sobib väga hästi Monte-Carlo ja kvaasi-Monte-Carlo meetoditega võrdlemiseks, kuna ta toimib praktiliselt sama skeemi järgi. Kuid ristkülikvalemi koondumine on üsna aeglane, seega kiirusele mõeldes tasuks kaaluda näiteks Simpsoni valemi kasutamist.

Kvadratuurvalemite taga on põhjalikult väljaarendatud teooria, muuhulgas on teada, et lähendi viga on suurusjärku $\mathrm{O}(n^{-2/d})$. Vea ligikaudseks hindamiseks iteratsioonide

käigus sobib aga Runge meetod, mille järgi on lähendi $\hat{I}_{km}$ viga antud juhul suurusjärku

$$\frac{\hat{I}_{km} - \hat{I}_m}{k^2 - 1}\,, \tag{3.1}$$

kus $\hat{I}_p$ on integraali lähend, kui servas on p punkti. Punktide arvu kasvu iseloomustava kordaja k väärtusteks esimese ja teise võrestikummetodi korral on vastavalt 2 ja 3. Runge veahinnang (3.1) töötab hästi, kui integreeritav funktsioon on piisavalt heade omadustega, kuid optsiooni maksefunktsiooni omadused ei ole piisavalt head.

Aga empiirilised katsetused näitasid, et optsiooni hindamise korral saab edukalt kasutada modifitseeritud Runge veahinnangut:

$$\frac{\hat{I}_{km} - \hat{I}_m}{k - 1}\,, \tag{3.2}$$

kus nimetajas on k^2 asemel k, mida võib interpreteerida oodatava koondumiskiiruse langusena antud kvadratuurvalemi jaoks.

4. Programmide detaile

4.1. Konfiguratsioon

Programmid on kirjutatud programmeerimiskeeles C. Erinevalt bakalaureusetööst ei kasutatud kompileerimiseks mitte Borlandi tarkvara, vaid DJGPP-nimelist kompilaatorit (versioon 2.03, vabavara, `http://www.delorie.com/djgpp/`). Üleminek on põhjustatud Borlandi kompilaatoris avastatud viperustest ning DJGPP-kompilaatori paremast ANSI-standardi järgimisest. Programme on jooksutatud järgmise konfiguratsiooniga arvutis: Intel Celeron 1.0 GHz, 512 MB RAM, Windows XP Professional.

4.2. Juhuslike arvude genereerimisest

Pseudojuhuslike ühtlase jaotusega arvude generaator on olemas valdavas enamuses programmeerimiskeskkondadest, kuid paljud neist on kahjuks nigelate omadustega. Näiteks realisatsiooniks võib olla valitud lineaarne kongruentne meetod ebaõnnestunud parameetritega.

Programeerimiskeele C standardseim pseudojuhuslike arvude generaator `rand()` on tavaliselt väga lühikese tsükliga — alla 33 tuhande, — seega ei ole rakendatav, kuna genereeritavate arvude hulk võib küündida kümnete miljoniteni. Teine, kuid juba mittestandardne funktsioon `drand48()` tundub olevat küll piisava tsüklipikkusega, kuid teised tema omadused ei ole kindlad.

Seepärast on kasutusele võetud üks väline pseudojuhuslike arvude generaator (sisuliselt kahe generaatori segu), väga heade omadustega ja ka üsna kiire. Realisatsiooni allikaks on tuntud arvuliste meetodite raamat [4].

4.3. Ühemõõtmelise normaaljaotuse modelleerimine

Kõigi vaadeldavate meetodite korral on mugav modelleerida normaaljaotust jaotusfunktsiooni pööramise teel, kuna just siis ühele ühtlase jaotusega suurusele vastab üks normaaljaotusega suurus (enamus teisi meetodeid seab vastavusse paarile kaks väärtust, lähemalt vt. [7]).

Jaotusfunktsiooni pööramise meetod kasutab asjaolu, et kui juhuslik suurus U on ühtlase jaotusega $\mathrm{U}(0,1)$ ning F on mingi jaotusfunktsioon, siis suurus $F^{-1}(U)$ on juhuslik suurus jaotusfunktsiooniga F.

Seega standardse normaaljaotuse modelleerimiseks peame pöörama selle jaotusfunktsiooni Φ. Kuigi analüütilist avaldist normaaljaotuse jaotusfunktsiooni pöördfunktsioonile ei leidu, saab kasutada mitmeid arvulisi lähendeid ning meetodeid. Käesoleva töö programmides on kasutatud üht lihtsamat varianti (vt. lähemalt [1]), mis on siiski väidetavalt üsna täpne — suhtelise vea absoluutväärtus on alla 1.15×10^{-9}.

Eeskiri standardse normaaljaotuse jaotusfunktsiooni pöördfunktsiooni arvutamiseks on järgmine:

$$\Phi^{-1}(x) = \begin{cases} \dfrac{c_1 q_1^5 + c_2 q_1^4 + c_3 q_1^3 + c_4 q_1^2 + c_5 q_1 + c_6}{d_1 q_1^4 + d_2 q_1^3 + d_3 q_1^2 + d_4 q_1 + 1}, & 0 < x < a, \\ \dfrac{(a_1 r_2^5 + a_2 r_2^4 + a_3 r_2^3 + a_4 r_2^2 + a_5 r_2 + a_6) q_2}{b_1 r_2^5 + b_2 r_2^4 + b_3 r_2^3 + b_4 r_2^2 + b_5 r_2 + 1}, & a \leqslant x \leqslant b, \\ -\dfrac{c_1 q_3^5 + c_2 q_3^4 + c_3 q_3^3 + c_4 q_3^2 + c_5 q_3 + c_6}{d_1 q_3^4 + d_2 q_3^3 + d_3 q_3^2 + d_4 q_3 + 1}, & b < x < 1, \end{cases}$$

kus

$$\begin{aligned} a &= 0.02425, & b &= 1-a\,, \\ q_1 &= \sqrt{-2\ln x}\,, & q_3 &= \sqrt{-2\ln(1-x)}\,, \\ q_2 &= x-0.5, & r_2 &= q_2^2 \end{aligned}$$

ning vajaminevate konstantide väärtusteks on

$$\begin{aligned} a_1 &= -3.969683028665376\times 10^1, & c_1 &= -7.784894002430293\times 10^{-3}, \\ a_2 &= 2.209460984245205\times 10^2, & c_2 &= -3.223964580411365\times 10^{-1}, \\ a_3 &= -2.759285104469687\times 10^2, & c_3 &= -2.400758277161838\times 10^0, \\ a_4 &= 1.383577518672690\times 10^2, & c_4 &= -2.549732539343734\times 10^0, \\ a_5 &= -3.066479806614716\times 10^1, & c_5 &= 4.374664141464968\times 10^0, \\ a_6 &= 2.506628277459239\times 10^0, & c_6 &= 2.938163982698783\times 10^0, \\ b_1 &= -5.447609879822406\times 10^1, & d_1 &= 7.784695709041462\times 10^{-3}, \\ b_2 &= 1.615858368580409\times 10^2, & d_2 &= 3.224671290700398\times 10^{-1}, \\ b_3 &= -1.556989798598866\times 10^2, & d_3 &= 2.445134137142996\times 10^0, \\ b_4 &= 6.680131188771972\times 10^1, & d_4 &= 3.754408661907416\times 10^0. \\ b_5 &= -1.328068155288572\times 10^1, \end{aligned}$$

4.4. Mitmemõõtmelise normaaljaotuse modelleerimine

Osates modelleerida ühemõõtmelist normaaljaotust, saame sisuliselt hakkama mitmemõõtmelise standardse normaaljaotuse $\mathrm{N}_n(\mathbf{0}, \boldsymbol{I})$ modelleerimisega. Teatud teisendusega saame lõpuks genereerida ka suvalise kovariatsioonimaatriksiga mitmemõõtmelise normaaljaotuse. Selleks kasutame normaaljaotuse järgmist omadust.

Teoreem 4.1. *Kui* $\boldsymbol{Y} \sim \mathrm{N}_n(\boldsymbol{\mu}, \boldsymbol{S})$, *siis*

$$\boldsymbol{Z} = \boldsymbol{A}\boldsymbol{Y} \sim \mathrm{N}_n\big(\boldsymbol{A}\boldsymbol{\mu}\,,\ \boldsymbol{A}\boldsymbol{S}\boldsymbol{A}^{\mathrm{T}}\big)\,,$$

kus $\boldsymbol{A}$ *on pööratav* $n \times n$*-maatriks.*

Tõestus on õpikus [7].

Teoreemist 4.1 saamegi idee, kuidas modelleerida vektorit $\boldsymbol{B}(T) \sim \mathrm{N}_n(\mathbf{0}, \boldsymbol{\Sigma})$ seosest (1.6). Esiteks genereerime juhusliku suuruse $\boldsymbol{B}_0 \sim \mathrm{N}_n(\mathbf{0}, \boldsymbol{I})$, mille eri koordinaadid on sõltumatud ühemõõtmelised standardsed normaaljaotused. Seejärel korrutame suurust $\boldsymbol{B}_0$ vasakult sobivalt valitud maatriksiga $\boldsymbol{A}$ (peab olema $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} = \boldsymbol{\Sigma}$). Siis

$$\boldsymbol{B}(T) = \boldsymbol{A}\boldsymbol{B}_0 \sim \mathrm{N}\big(\boldsymbol{A}\mathbf{0}, \boldsymbol{A}\boldsymbol{I}\boldsymbol{A}^{\mathrm{T}}\big) = \mathrm{N}(\mathbf{0}, \boldsymbol{\Sigma}).$$

Kuna maatriks $\boldsymbol{\Sigma}$ on sümmeetriline ja positiivselt määratud, siis selline maatriks $\boldsymbol{A}$ on kindlasti olemas. Probleem seisneb vaid tema leidmises. Üheks võimaluseks on kasutada maatriksi $\boldsymbol{\Sigma}$ omaväärtusi ja omavektoreid. Teine võimalus maatriksi $\boldsymbol{A}$ leidmiseks on kolmnurkse maatriksi kasutamine. Nimelt saab näidata, et iga positiivselt määratud maatriksi $\boldsymbol{\Sigma}$ korral leidub positiivsete diagonaalielementidega alumine kolmnurkne maatriks $\boldsymbol{A}$ nii, et $\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} = \boldsymbol{\Sigma}$. Näiteks juhul $n = 3$ oleks maatriks $\boldsymbol{A}$ järgmise struktuuriga:

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$

Maatriksi $\boldsymbol{A}$ elemendid a_{ij} saame leida üksteise järel, lähtudes tingimusest

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{pmatrix} = \begin{pmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ 0 & a_{22} & a_{32} \\ 0 & 0 & a_{33} \end{pmatrix} = \boldsymbol{A}\boldsymbol{A}^{\mathrm{T}},$$

kust

$$\begin{aligned}
&\sigma_{11} = a_{11}^2 && \Rightarrow && a_{11} = \sqrt{\sigma_{11}}\,, \\
&\sigma_{12} = a_{11}a_{21} && \Rightarrow && a_{21} = \frac{\sigma_{12}}{a_{11}}\,, \\
&\sigma_{22} = a_{21}^2 + a_{22}^2 && \Rightarrow && a_{22} = \sqrt{\sigma_{22} - a_{21}^2}\,, \\
&\sigma_{13} = a_{11}a_{31} && \Rightarrow && a_{31} = \frac{\sigma_{13}}{a_{11}}\,, \\
&\sigma_{23} = a_{21}a_{31} + a_{22}a_{32} && \Rightarrow && a_{32} = \frac{\sigma_{23} - a_{21}a_{31}}{a_{22}}\,, \\
&\sigma_{33} = a_{31}^2 + a_{32}^2 + a_{33}^2 && \Rightarrow && a_{33} = \sqrt{\sigma_{33} - a_{31}^2 - a_{32}^2}\,.
\end{aligned}$$

Kahemõõtmelisel juhul elementide a_{11} , a_{21} ning a_{22} avaldised jäävad samaks.

4.5. Programmidest

Lisades A ning B toodud programmide lähtetekstid on lisaks salvestatud laserplaadi peale (lisa C). Huvitatud uurija võib neid programme kompileerida (nt. samal laserplaadil oleva kompilaatori abil) ning jooksutada. Programmides saab näiteks muuta väärtpaberituru parameetreid ning maksefunktsiooni.

Optsiooni hinna arvutamise käigus annavad programmid informatsiooni jooksva seisu kohta, sh. näidates hinnalähendit ja kulutatud aega. Samuti on programmides olemas lõigud, mis lubavad salvestada koondumisinformatsiooni eraldi faili.

5. Arvutustulemused ja võrdlused

5.1. Kahe aktsia juht

Kahemõõtmelisel juhul teeme arvutusi järgmiste parameetrite väärtustega:

ε	α	S_0	K	T	r	σ
0.01	0.01	$\begin{pmatrix}40\\60\end{pmatrix}$	$\begin{pmatrix}50\\50\end{pmatrix}$	0.5	0.09531	$\begin{pmatrix}0.3 & 0\\0 & 0.3\end{pmatrix}$

Tabel 5.1. Kahemõõtmelise testimise arvulised parameetrid.

Siin ε on soovitav täpsus, α on maksimaalne lubatav eksimistõenäosus, $S_0 = \left(\begin{smallmatrix}S_1(0)\\S_2(0)\end{smallmatrix}\right)$ on alusväärtpaberite hindade vektor hetkel $t = 0$, $K = \left(\begin{smallmatrix}K_1\\K_2\end{smallmatrix}\right)$ on täitmishindade vektor, T on optsiooni eluiga, r on pangaprotsent, $\sigma = \left(\begin{smallmatrix}\sigma_{11} & \sigma_{12}\\\sigma_{21} & \sigma_{22}\end{smallmatrix}\right)$ on volatiilsuste maatriks.

5.1.1. Ostuoptsioon

Maksefunktsiooniks valime kahemõõtmelise Euroopa tüüpi ostuoptsiooni maksefunktsiooni:

$$p\big(S_1, S_2\big) = \max\{0, S_1 - K_1, S_2 - K_2\}\,, \tag{5.1}$$

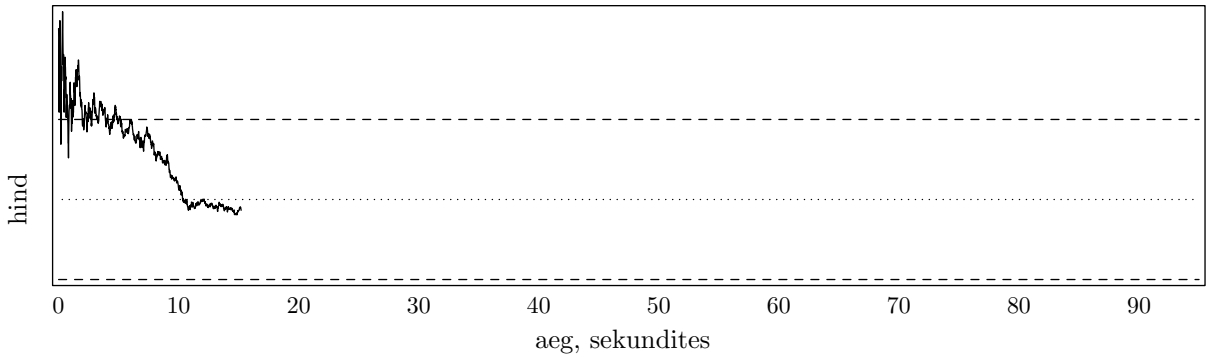
kus $S_i = S_i(T)$ on alusväärtpaberite hinnad täitmisajal ja K_i on vastavad täitmishinnad. Täpne hind antud parameetrite korral on 13.471145 (arvutatakse R. Kangro poolt kirjutatud programmi abil matemaatilises paketis Maple).

Eri meetoditel kulus etteantud täpsuse saavutamiseks ka erinevalt aega. Keskeltläbi olid need ajad sellised:

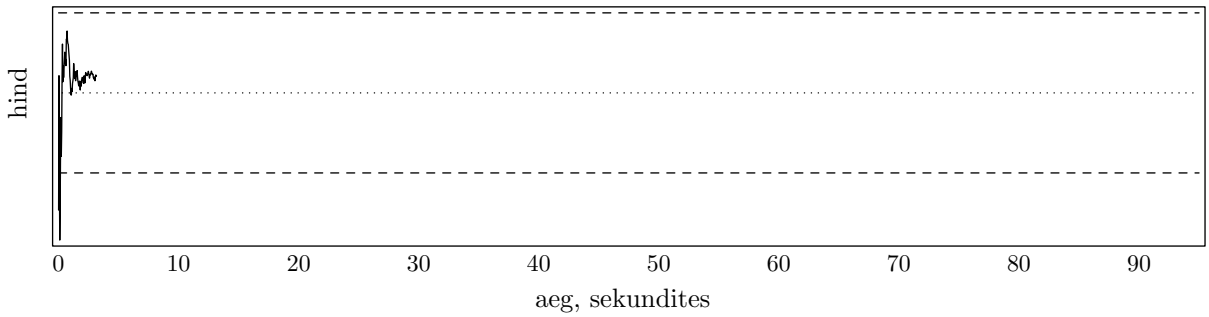
Meetod	Aeg (s)	Veahinnang
Monte-Carlo	16	normaaljaotusega lähendamine
Monte-Carlo, antiteetiline	3	normaaljaotusega lähendamine
kvaasi-Monte-Carlo, Haltoni jada	4	«silma järgi»
kvaasi-Monte-Carlo, Soboli jada	0.5	«silma järgi»
esimene võrestikumeetod	0.4	modifitseeritud Runge meetod
teine võrestikumeetod	0.2	modifitseeritud Runge meetod

Tabel 5.2. Eri meetodite ajad kahemõõtmelise ostuoptsiooni hinna arvutamiseks.

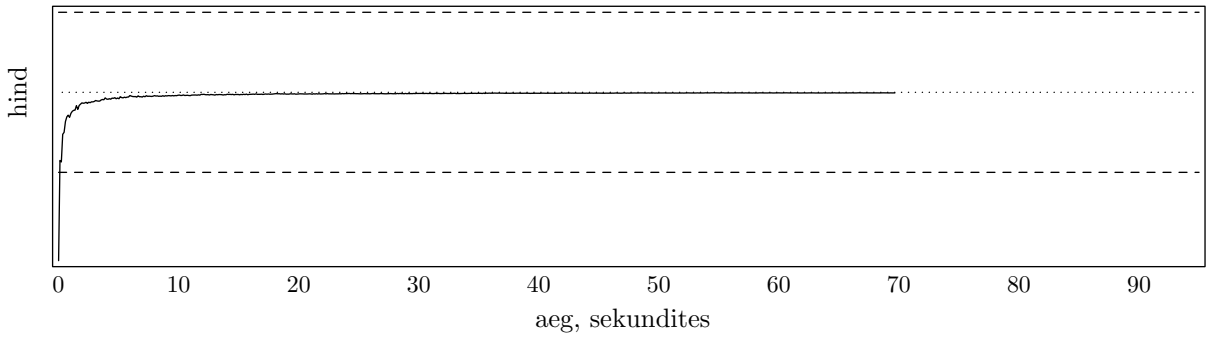
Järgnevad joonised annavad ettekujutuse, kuidas käitus hinnalähend eri meetodite koondumise ajal. Punktjoon tähistab optsiooni täpset hinda, kaks katkendlikku joont määravad nõutava täpsuse koridori, pidev joon esitab hinnalähendit.



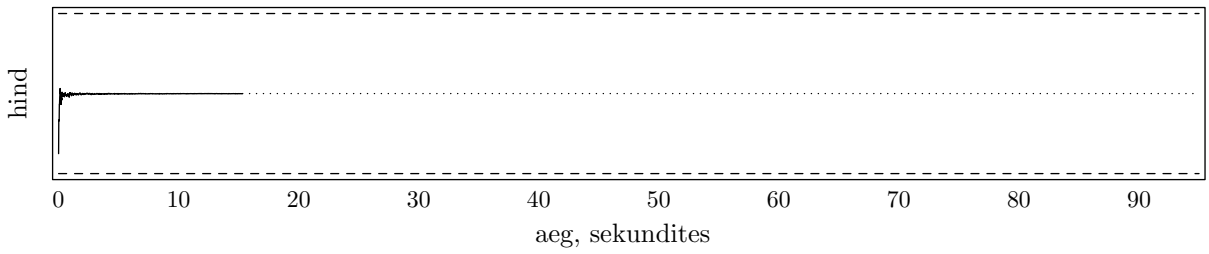
Joonis 5.1. Kahemõõtmelise ostuoptsiooni hinnalähend Monte-Carlo meetodi korral.



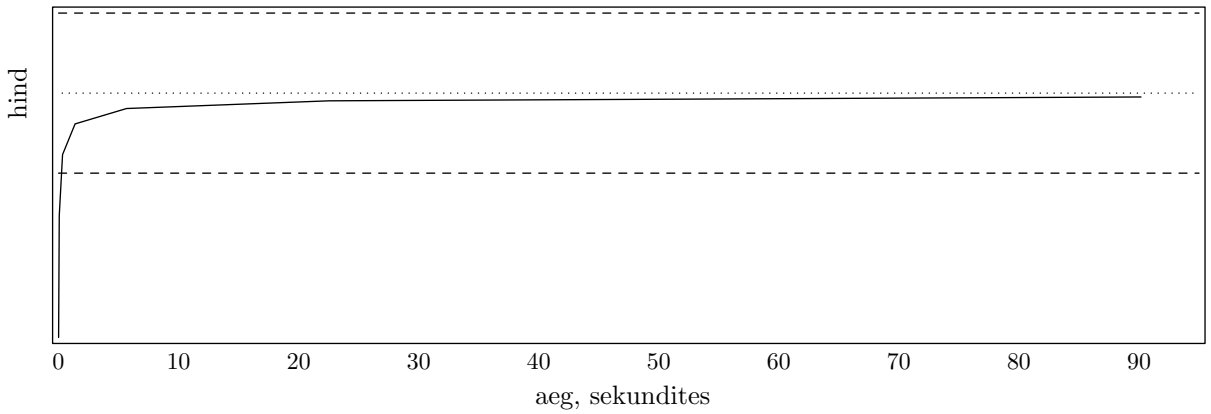
Joonis 5.2. Kahemõõtmelise ostuoptsiooni hinnalähend antiteetilise Monte-Carlo meetodi korral.



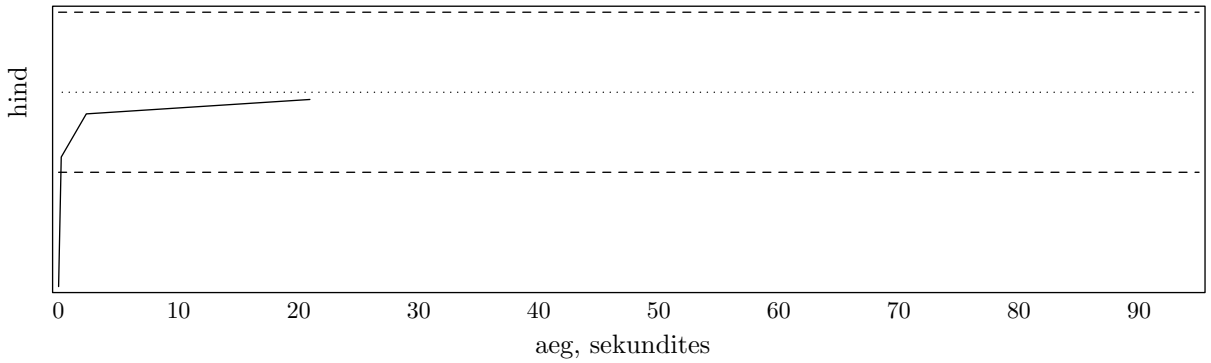
Joonis 5.3. Kahemõõtmelise ostuoptsiooni hinnalähend kvaasi-Monte-Carlo meetodi korral (Haltoni jada).



Joonis 5.4. Kahemõõtmelise ostuoptsiooni hinnalähend kvaasi-Monte-Carlo meetodi korral (Soboli jada).



Joonis 5.5. Kahemõõtmelise ostuoptsiooni hinnalähend esimese võrestikumeetodi korral.



Joonis 5.6. Kahemõõtmelise ostuoptsiooni hinnalähend teise võrestikumeetodi korral.

Ära kulub ka selgitus, miks lõpevad hinnalähendi jooned eri meetodite korral eri aegadel. Monte-Carlo ja antiteetilise Monte-Carlo meetodi korral on põhjuseks veahinnangu nõude rahuldamine; kvaasi-Monte-Carlo meetoditel oli lubatud töötada vajalikust kauem, et saada selgema pilti hinnalähendi käitumisest; võrestikumeetodite tulemustest on joonisele kantud kõik, mis mahtus pooleteise minuti sisse. Lõppkokkuvõttes annab see suurema ülevaatlikuse.

Kahe võrestikumeetodi illustreerimiseks toome ära ka iteratsioonide tabelid.

Samm	Punktide koguarv	Hinnalähend	Modifitseeritud Runge veahinnang	Tegelik viga	Aeg (s)
1	4	11.593470	11.593470	1.877676	0.000000
2	16	12.504797	0.911328	0.966348	0.000000
3	64	12.993857	0.489060	0.477288	0.000000
4	256	13.228157	0.234300	0.242988	0.000000
5	1 024	13.350444	0.122287	0.120701	0.000000
6	4 096	13.409617	0.059173	0.061528	0.000000
7	16 384	13.440616	0.030999	0.030530	0.054945
8	65 536	13.455824	0.015209	0.015321	0.109890
9	262 144	13.463464	0.007639	0.007682	0.384615
10	1 048 576	13.467298	0.003835	0.003847	1.428571
11	4 194 304	13.469220	0.001922	0.001925	5.769231
12	16 777 216	13.470182	0.000962	0.000964	23.186813
13	67 108 864	13.470663	0.000481	0.000482	92.307692
14	268 435 456	13.470904	0.000241	0.000242	381.483516

Tabel 5.3. Esimene võrestikumeetod kahemõõtmelise ostuoptsiooni korral.

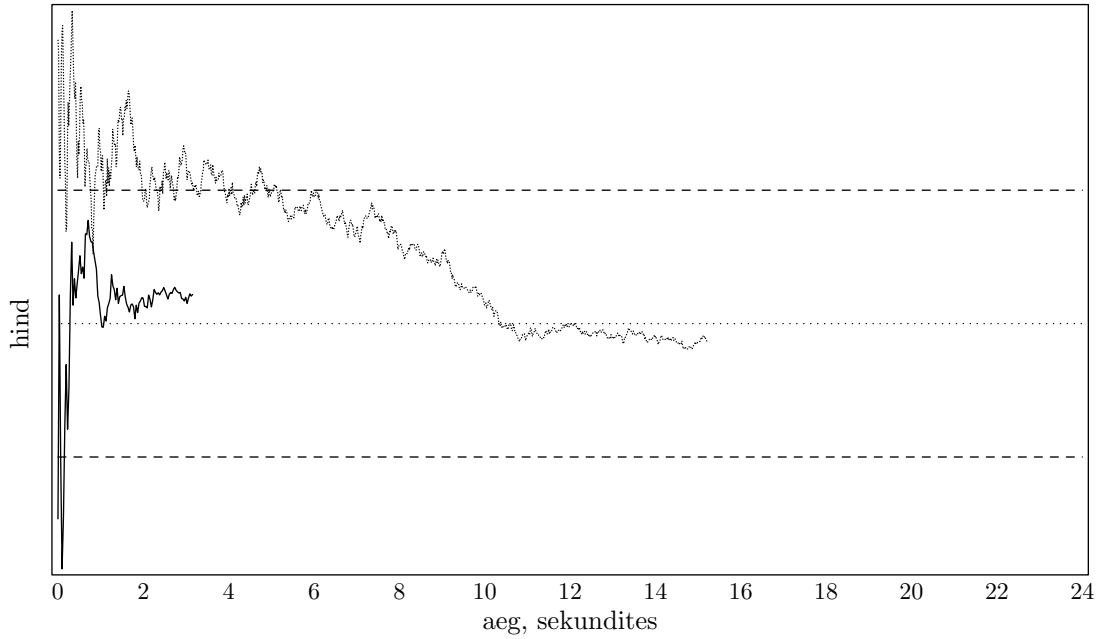
Samm	Punktide koguarv	Hinnalähend	Modifitseeritud Runge veahinnang	Tegelik viga	Aeg (s)
1	4	11.593470	5.796735	1.877676	0.000000
2	36	12.890577	0.648553	0.580569	0.000000
3	324	13.262744	0.186084	0.208401	0.000000
4	2 916	13.399345	0.068300	0.071801	0.000000
5	26 244	13.446866	0.023761	0.024280	0.000000
6	236 196	13.463048	0.008091	0.008097	0.219780
7	2 125 764	13.468442	0.002697	0.002703	2.307692
8	19 131 876	13.470243	0.000900	0.000903	20.769231
9	172 186 884	13.470844	0.000301	0.000301	186.318681

Tabel 5.4. Teine võrestikumeetod kahemõõtmelise ostuoptsiooni korral.

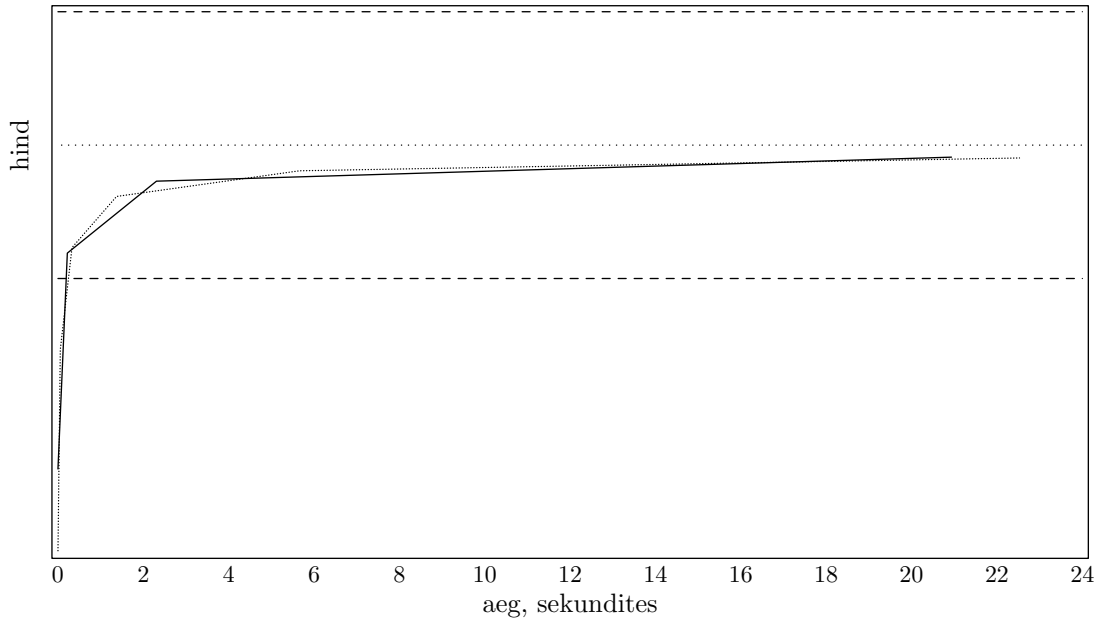
Esimesed järeldused on järgmised:

- Nagu oligi oodata, antiteetiline Monte-Carlo on tavalisest Monte-Carlo meetodist tunduvalt kiirem; saadud kiirendus on umbes viiekordne.
- Mõlemad kvaasi-Monte-Carlo meetodid on Monte-Carlo meetodist oluliselt kiiremad, see on tegelikult oodatav. Seejuures Soboli meetod edestab Haltoni meetodit, ning selle taga on kindlasti asjaolu, et Soboli meetodit realiseeriv funktsioon on Haltoni meetodi omast kiiremini arvutatav; ei ole välistatud, et Soboli jada omadused on samuti paremad.
- Antiteetilise Monte-Carlo ning Haltoni meetodite tööajad on väga lähedased; siiski tasub pidada silmas, et kvaasi-Monte-Carlo meetodite veahinnang on üsna jäme.
- Mõlemad võrestikumeetodid on väga kiired (võrreldavad Soboli algoritmiga). Modifitseeritud Runge meetodil põhinev veahinnang töötab laitmatult. Aga nagu oligi karta, saabub üsna kiiresti hetk, kui järgmist iteratsiooni lihtsalt ei jõua ära oodata.

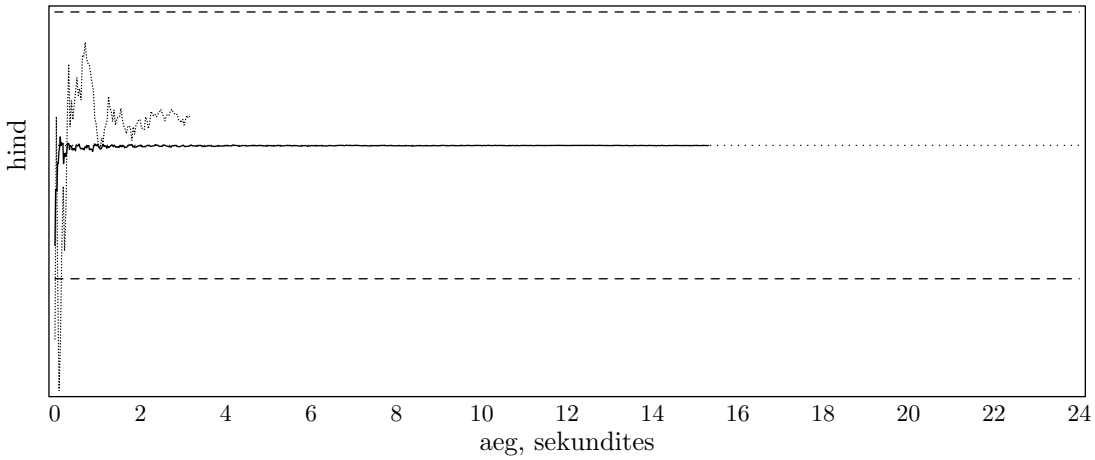
Kindlasti pakuvad huvi ka detailsemad omavahelised meetodite võrdlused. Järgnevalt on esitatud valik eri meetodite paaridest.



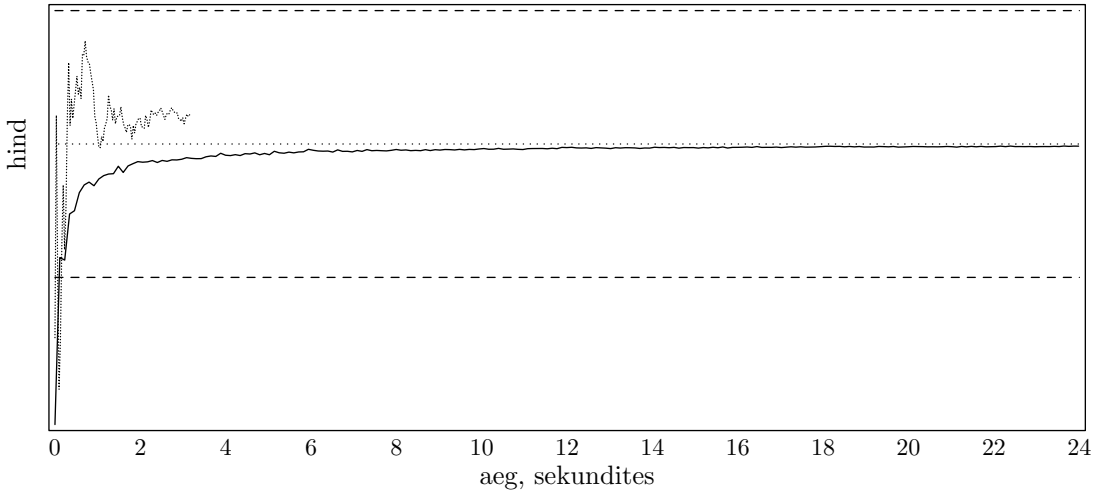
Joonis 5.7. Kahemõõtmelise ostuoptsiooni hinnalähendid Monte-Carlo (punktjoon) ja antiteetilise Monte-Carlo meetodi korral.



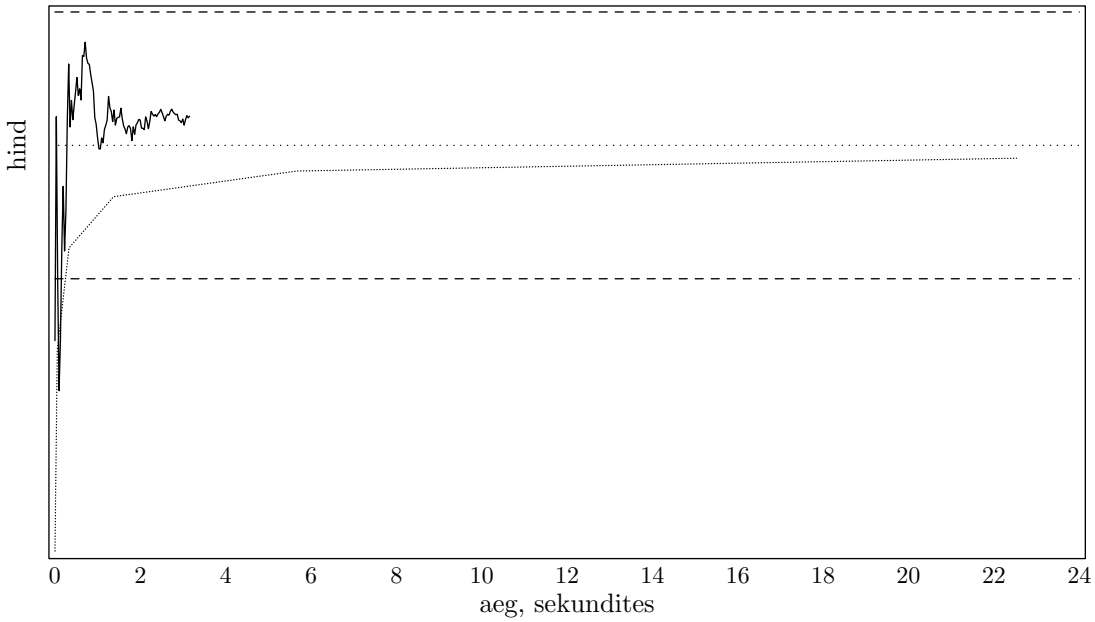
Joonis 5.8. Kahemõõtmelise ostuoptsiooni hinnalähendid esimese (punktjoon) ja teise võrestikumeetodi korral.



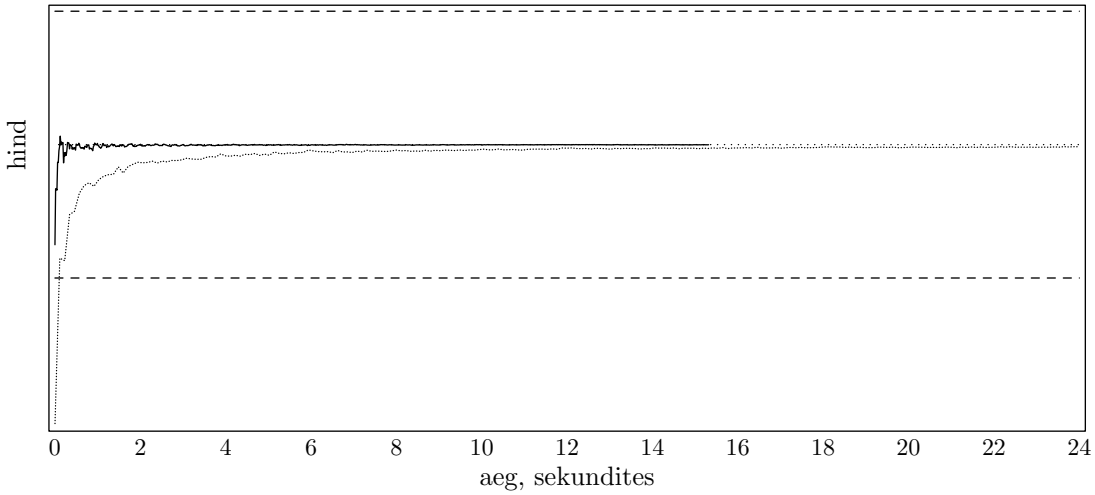
Joonis 5.9. Kahemõõtmelise ostuoptsiooni hinnalähendid antiteetilise Monte-Carlo (punktjoon) ja Soboli meetodi korral.



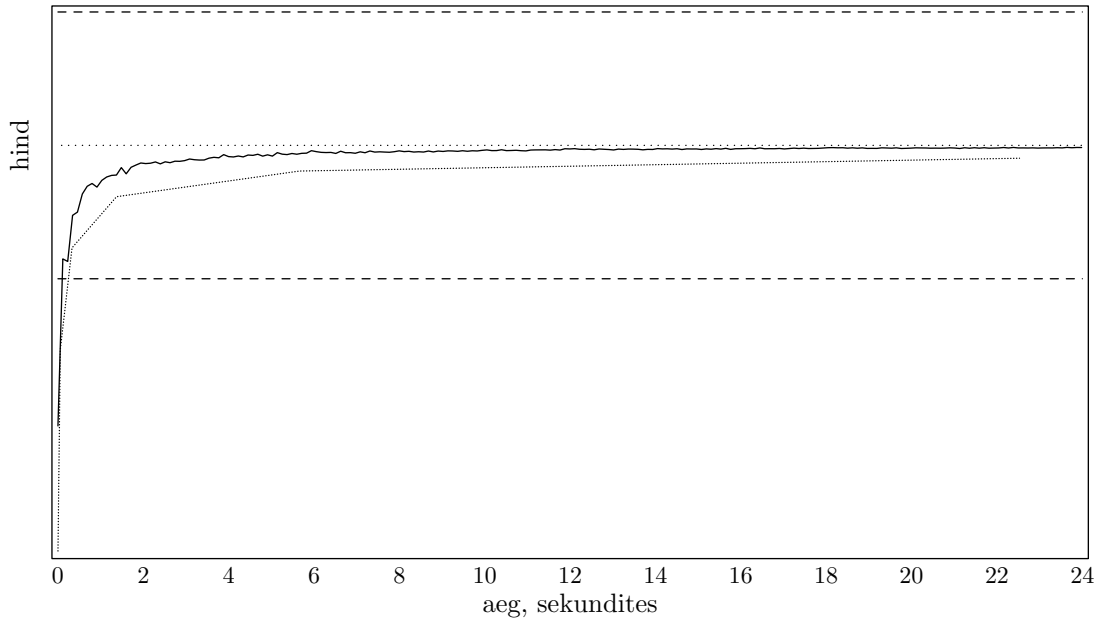
Joonis 5.10. Kahemõõtmelise ostuoptsiooni hinnalähendid antiteetilise Monte-Carlo (punktjoon) ja Haltoni meetodi korral.



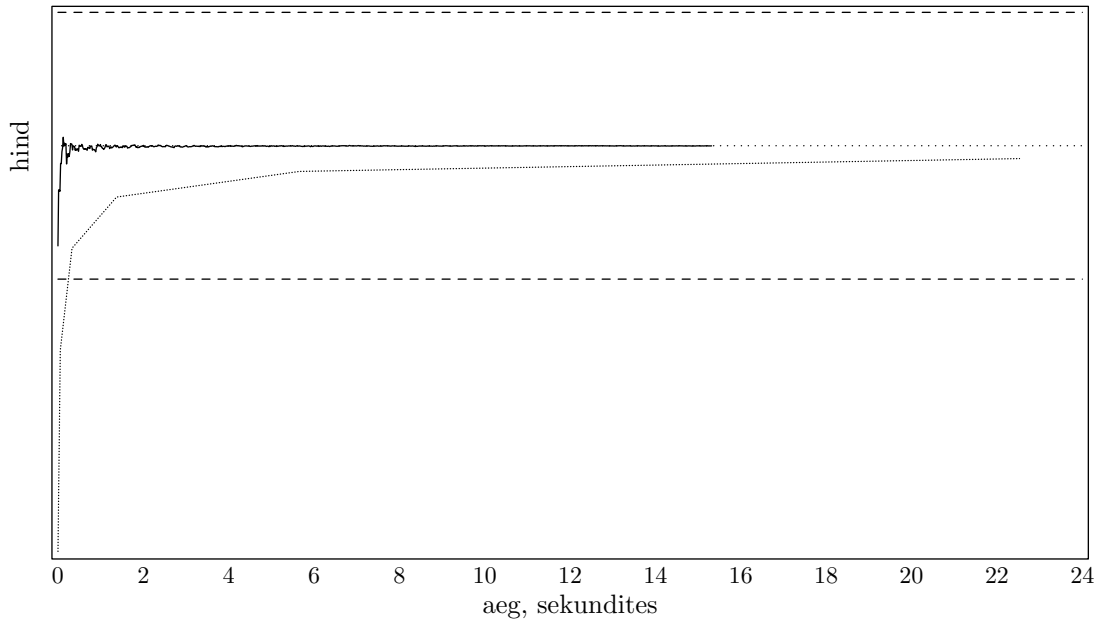
Joonis 5.11. Kahemõõtmelise ostuoptsiooni hinnalähendid esimese võrestikumeetodi (punktjoon) ja antiteetilise Monte-Carlo meetodi korral.



Joonis 5.12. Kahemõõtmelise ostuoptsiooni hinnalähend Haltoni (punktjoon) ja Soboli meetodi korral.



Joonis 5.13. Kahemõõtmelise ostuoptsiooni hinnalähend esimese võrestikumeetodi (punktjoon) ja Haltoni meetodi korral.



Joonis 5.14. Kahemõõtmelise ostuoptsiooni hinnalähend esimese võrestikumeetodi (punktjoon) ja Soboli meetodi korral.

Paarikaupa joonistest on näha, et

- kaks vaadeldud võrestikumeetodit ei erine üksteisest tähelepanuväärselt;
- kuigi etteantud täpsuse saavutamine toimub võrestiku- ja kvaasi-Monte-Carlo meetodite korral enam-vähem üheaegselt, on kvaasi-Monte-Carlo meetodid omaette kõrgema täpsuseklassiga;
- tavaline Monte-Carlo meetod jääb selgelt teistele meetoditele alla;

- antiteetiline Monte-Carlo meetod ei jää kuigi palju maha võrestikumeetoditest;
- Soboli algoritmil põhinev meetod selgelt edestab teisi meetodeid.

5.1.2. Müügioptsioon

Maksefunktsiooniks antud juhul valime kahemõõtmelise Euroopa tüüpi müügioptsiooni maksefunktsiooni:

$$p\big(S_1, S_2\big) = \max\{0, K_1 - S_1, K_2 - S_2\}\,, \tag{5.2}$$

kus $S_i = S_i(T)$ on alusväärtpaberite hinnad täitmisajal ja K_i on vastavad täitmishinnad. Müügioptsiooni maksefunktsioon (5.2) on teatud mõttes ostuoptsiooni maksefunktsiooni (5.1) «peegelduseks», ning maksefunktsiooni valik võib mõjutada mõne meetodi koondumiskiirust.

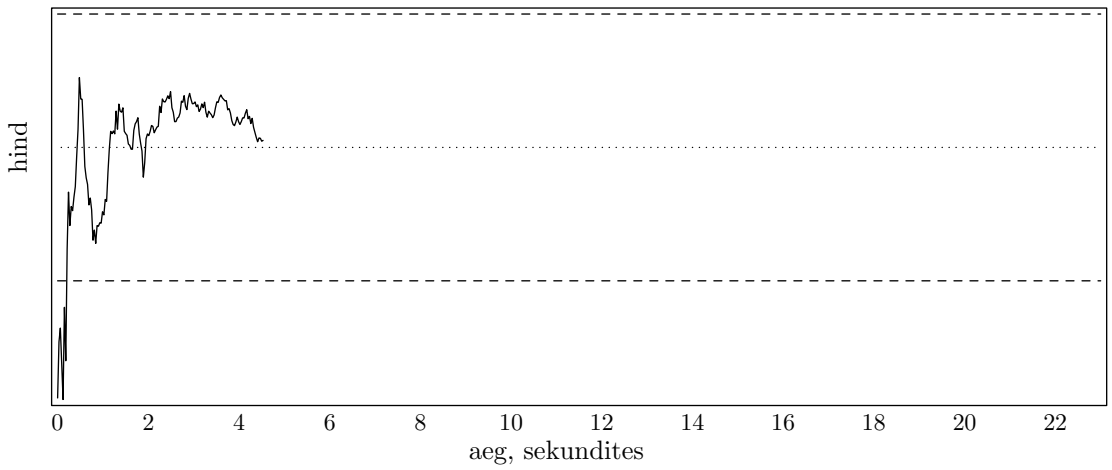
Täpseks hinnaks antud juhul on 8.98100. Eri meetodite tööajad on esitatud järgnevas tabelis.

Meetod	Aeg (s)
Monte-Carlo	5
Monte-Carlo, antiteetiline	0.4
kvaasi-Monte-Carlo, Haltoni jada	0.5
kvaasi-Monte-Carlo, Soboli jada	0.1
esimene võrestikumeetod	0.1
teine võrestikumeetod	0.2

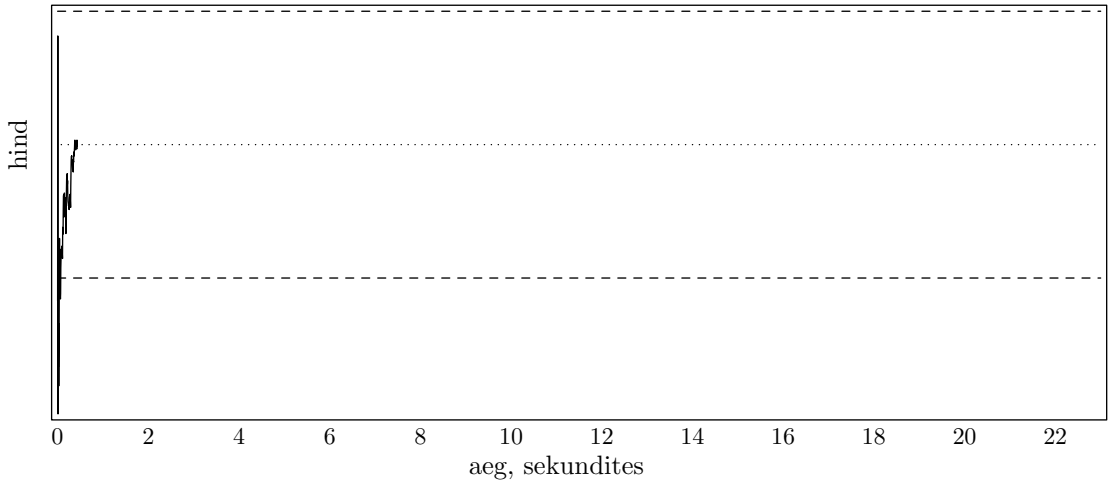
Tabel 5.5. Eri meetodite ajad kahemõõtmelise müügioptsiooni hinna arvutamiseks.

Endiselt on antiteetilise Monte-Carlo ja Haltoni meetodite ajad väga lähedased. Ning võrestikumeetodid koos Soboli algoritmiga on endiselt koos ja teistest ees. Antiteetiline Monte-Carlo meetod on nüüd tavalisest Monte-Carlo meetodist ligi 10 korda kiirem.

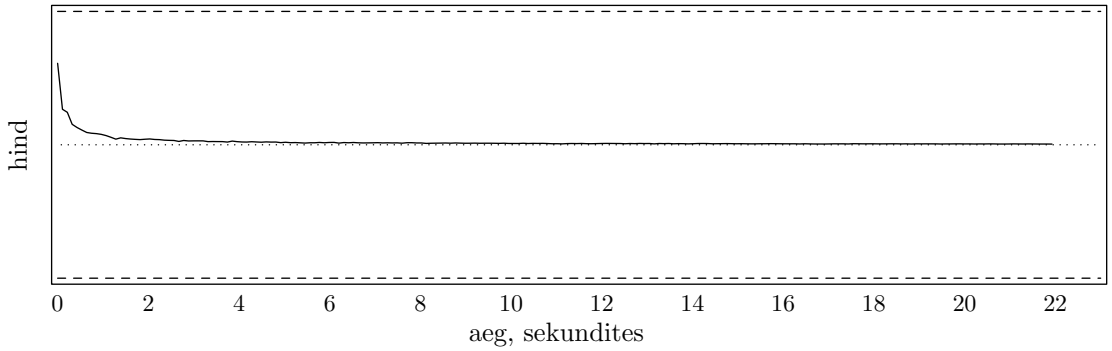
Antud juhul käituvad hinnalähendid järgmiselt.



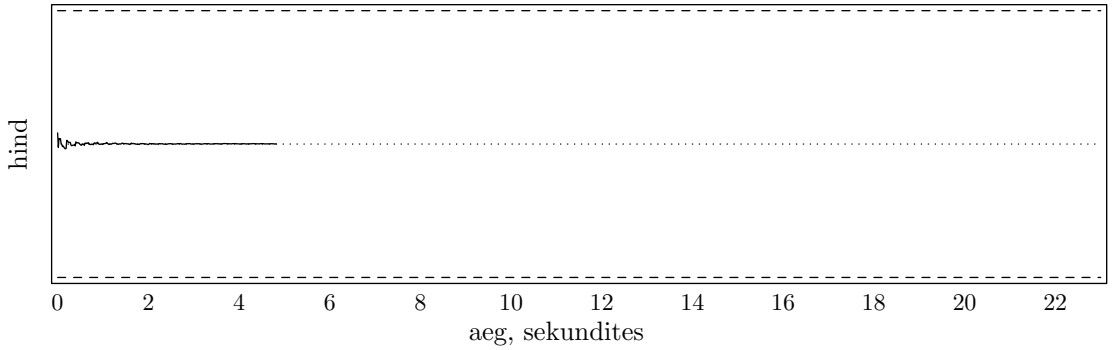
Joonis 5.15. Kahemõõtmelise müügioptsiooni hinnalähend Monte-Carlo meetodi korral.



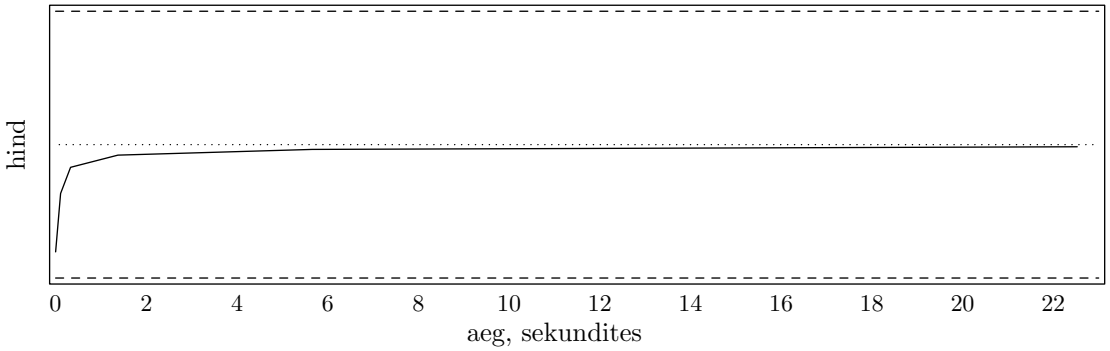
Joonis 5.16. Kahemõõtmelise müügioptsiooni hinnalähend antiteetilise Monte-Carlo meetodi korral.



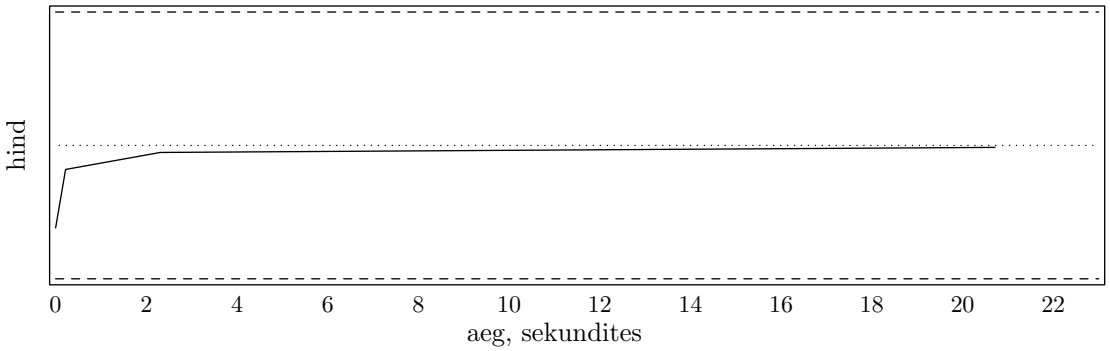
Joonis 5.17. Kahemõõtmelise müügioptsiooni hinnalähend kvaasi-Monte-Carlo meetodi korral (Haltoni jada).



Joonis 5.18. Kahemõõtmelise müügioptsiooni hinnalähend kvaasi-Monte-Carlo meetodi korral (Soboli jada).



Joonis 5.19. Kahemõõtmelise müügioptsiooni hinnalähend esimese võrestikumeetodi korral.



Joonis 5.20. Kahemõõtmelise müügioptsiooni hinnalähend teise võrestikumeetodi korral.

Järgnevalt esitame võrestikumeetodite iteratsioonide tabelid.

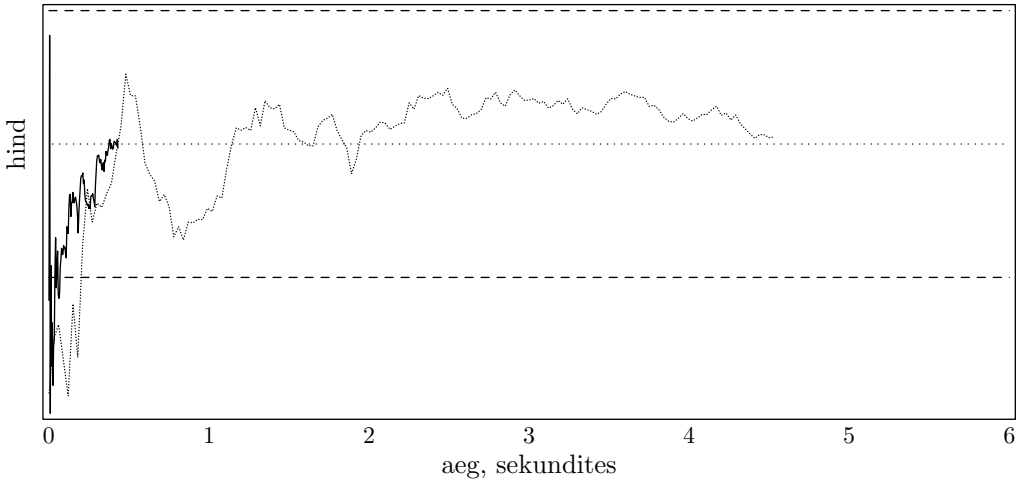
Samm	Punktide koguarv	Hinnalähend	Modifitseeritud Runge veahinnang	Tegelik viga	Aeg (s)
1	4	8.162063	8.162063	0.818937	0.000000
2	16	8.601707	0.439643	0.379293	0.000000
3	64	8.793380	0.191673	0.187620	0.000000
4	256	8.903481	0.110101	0.077519	0.000000
5	1024	8.943149	0.039667	0.037851	0.000000
6	4096	8.963880	0.020731	0.017120	0.000000
7	16384	8.972960	0.009080	0.008040	0.000000
8	65536	8.977339	0.004379	0.003661	0.109890
9	262144	8.979299	0.001960	0.001701	0.329670
10	1048576	8.980213	0.000914	0.000787	1.373626
11	4194304	8.980642	0.000429	0.000358	5.659341
12	16777216	8.980840	0.000198	0.000160	22.527473
13	67108864	8.980933	0.000093	0.000067	90.219780
14	268435456	8.980976	0.000043	0.000024	360.769231

Tabel 5.6. Esimene võrestikumeetod kahemõõtmelise müügioptsiooni korral.

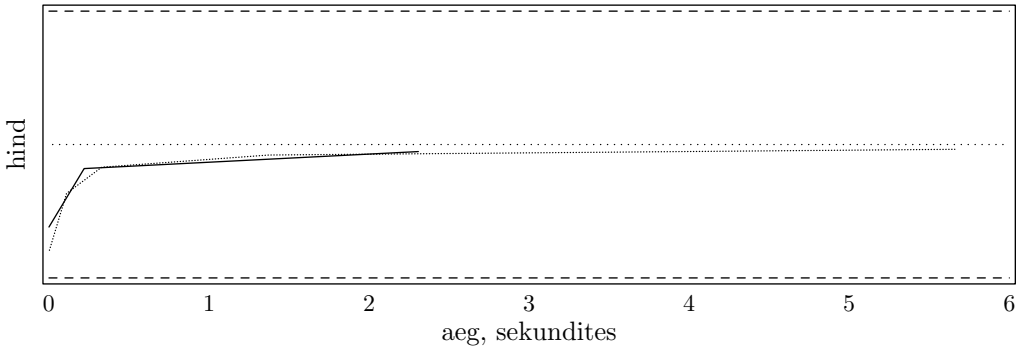
Samm	Punktide koguarv	Hinnalähend	Modifitseeritud Runge veahinnang	Tegelik viga	Aeg (s)
1	4	8.162063	4.081032	0.818937	0.000000
2	36	8.793846	0.315891	0.187154	0.000000
3	324	8.915376	0.060765	0.065624	0.000000
4	2916	8.959681	0.022153	0.021319	0.000000
5	26244	8.974816	0.007568	0.006184	0.000000
6	236196	8.979197	0.002190	0.001803	0.219780
7	2125764	8.980472	0.000638	0.000528	2.307692
8	19131876	8.980852	0.000190	0.000148	20.714286
9	172186884	8.980966	0.000057	0.000034	186.373626

Tabel 5.7. Teine võrestikumeetod kahemõõtmelise müügioptsiooni korral.

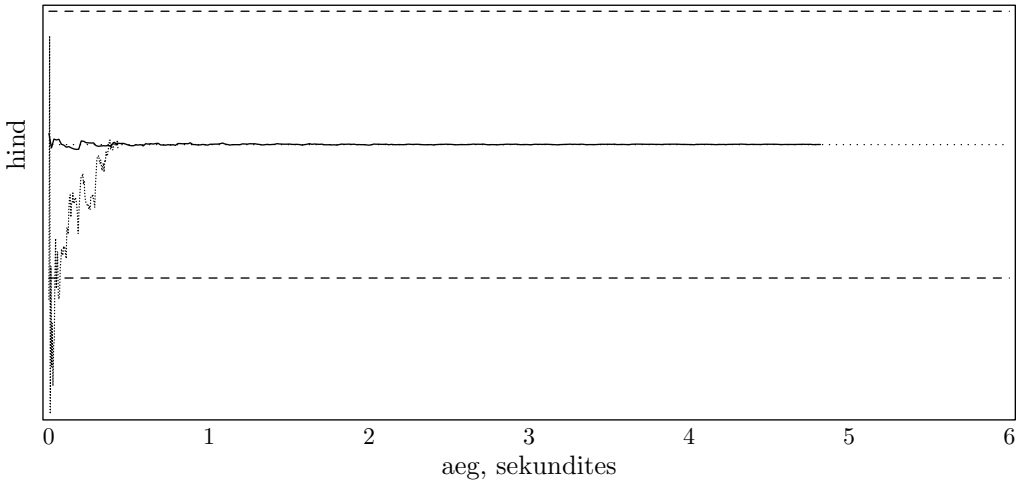
Lõpuks esitame joonised eri meetodite kõige huvitavamate paaride kohta.



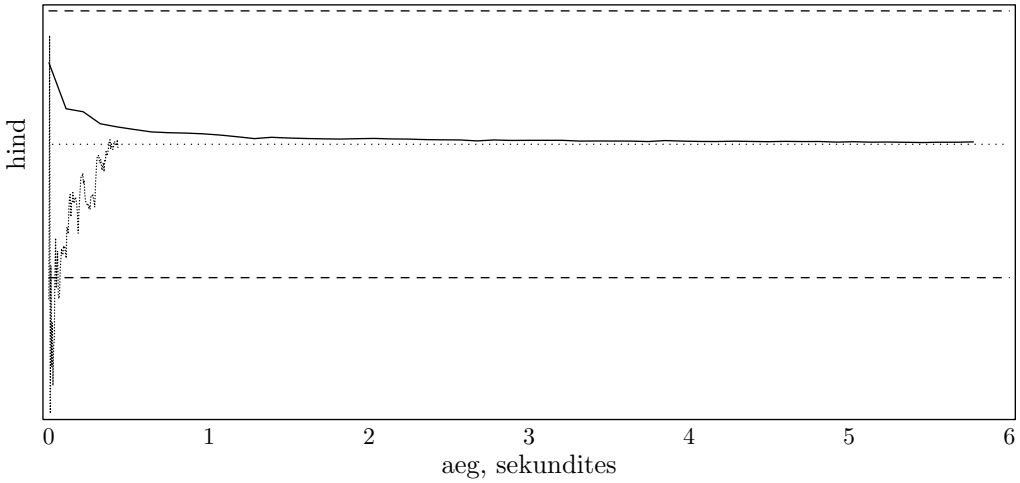
Joonis 5.21. Kahemõõtmelise müügioptsiooni hinnalähendid Monte-Carlo (punktjoon) ja antiteetilise Monte-Carlo meetodi korral.



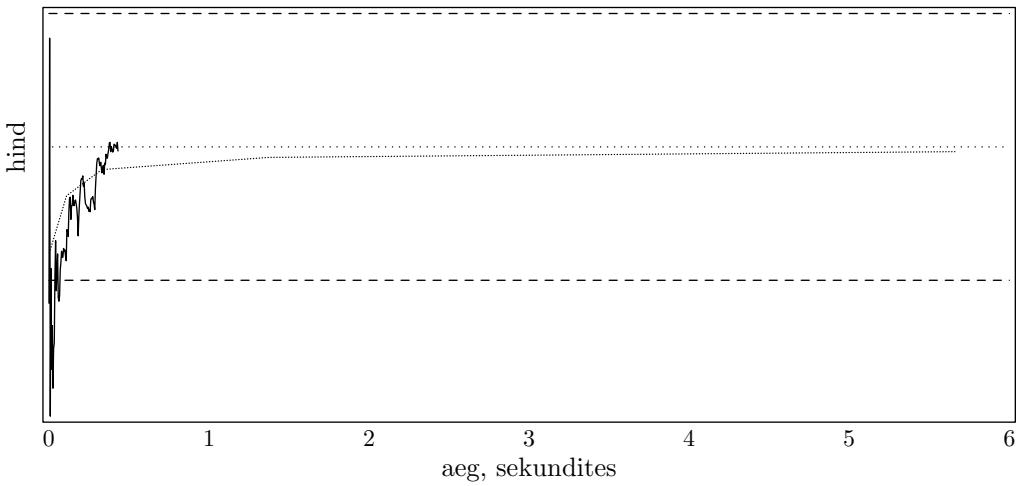
Joonis 5.22. Kahemõõtmelise müügioptsiooni hinnalähendid esimese (punktjoon) ja teise võrestikumeetodi korral.



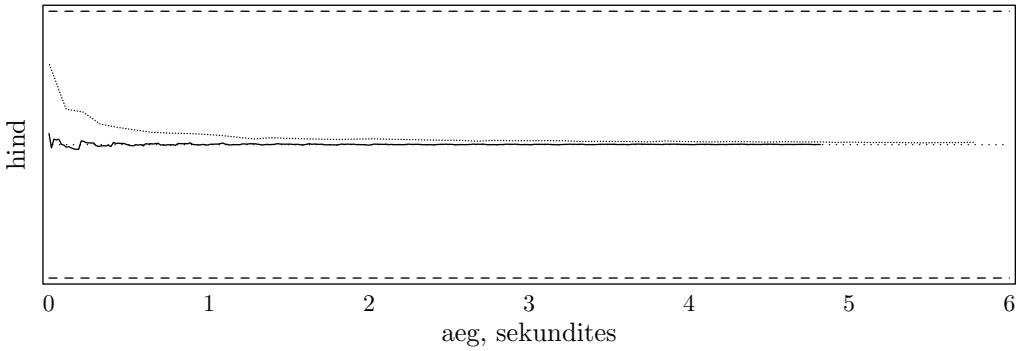
Joonis 5.23. Kahemõõtmelise müügioptsiooni hinnalähendid antiteetilise Monte-Carlo (punktjoon) ja Soboli meetodi korral.



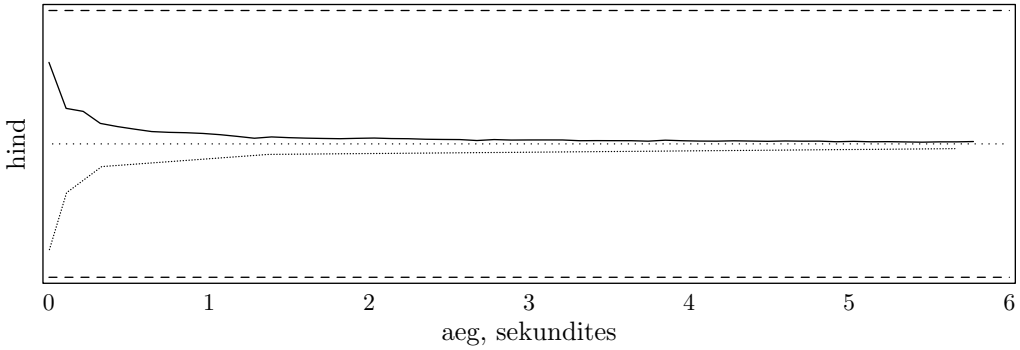
Joonis 5.24. Kahemõõtmelise müügioptsiooni hinnalähendid antiteetilise Monte-Carlo (punktjoon) ja Haltoni meetodi korral.



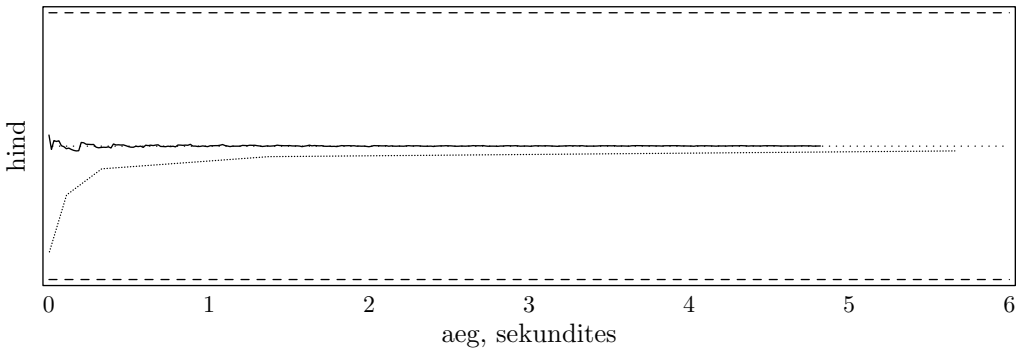
Joonis 5.25. Kahemõõtmelise müügioptsiooni hinnalähendid esimese võrestikumeetodi (punktjoon) ja antiteetilise Monte-Carlo meetodi korral.



Joonis 5.26. Kahemõõtmelise müügioptsiooni hinnalähend Haltoni (punktjoon) ja Soboli meetodi korral.



Joonis 5.27. Kahemõõtmelise müügioptsiooni hinnalähend esimese võrestikumeetodi (punktjoon) ja Haltoni meetodi korral.



Joonis 5.28. Kahemõõtmelise müügioptsiooni hinnalähend esimese võrestikumeetodi (punktjoon) ja Soboli meetodi korral.

Muutusi meetodite vahekorras tegelikult ei ole, esile võiks tuua vaid seda, et väiksema hinna tõttu koonduvad kõik meetodid märgatavalt kiiremini.

5.2. Kolme aktsia juht

Kolmemõõtmeline juhtum pakub siinkohal huvi, kuna selle tulemuste põhjal saab teha järeldusi, kuidas käituvad vaadeldavad meetodid dimensiooni kasvades. Parameetrite väärtused on valitud vastavalt tabelile 5.8.

ε	α	S_0	K	T	r	σ
0.01	0.01	$\begin{pmatrix}40\\60\\50\end{pmatrix}$	$\begin{pmatrix}50\\50\\50\end{pmatrix}$	0.5	0.09531	$\begin{pmatrix}0.3 & 0 & 0.1\\0 & 0.3 & 0.1\\0.1 & 0.1 & 0.2\end{pmatrix}$

Tabel 5.8. Kolmemõõtmelise testimise arvulised parameetrid.

5.2.1. Ostuoptsioon

Maksefunktsiooniks on kolmemõõtmelise Euroopa tüüpi ostuoptsiooni maksefunktsioon:

$$p\big(S_1, S_2, S_3\big) = \max\{0, S_1 - K_1, S_2 - K_2, S_3 - K_3\}\,.$$

Täpne hind antud parameetrite korral on 13.87329. Eri meetodite tööajad on esitatud järgnevas tabelis.

Meetod	Aeg (s)
Monte-Carlo	25
Monte-Carlo, antiteetiline	11
kvaasi-Monte-Carlo, Haltoni jada	6
kvaasi-Monte-Carlo, Soboli jada	1
esimene võrestikumeetod	240
teine võrestikumeetod	190

Tabel 5.9. Eri meetodite ajad kolmemõõtmelise ostuoptsiooni hinna arvutamiseks.

Kvaasi-Monte-Carlo meetodid on selgelt paremad, nad töötavad endiselt täiesti mõistliku ajaga. Antiteetiline Monte-Carlo meetod on tavalisest Monte-Carlo meetodist umbes kaks korda kiirem. Võrestikumeetoditel esineb aga probleeme järjekordse iteratsioonisammu lõpetamisega, tabelid iteratsioonide illustreerimiseks on seekord järgmised:

Samm	Punktide koguarv	Hinnalähend	Modifitseeritud Runge veahinnang	Tegelik viga	Aeg (s)
1	8	11.689747	11.689747	2.183543	0.000000
2	64	12.839604	1.149857	1.033686	0.000000
3	512	13.327835	0.488231	0.545455	0.000000
4	4096	13.606431	0.278596	0.266859	0.000000
5	32768	13.737396	0.130965	0.135894	0.054945
6	262144	13.804523	0.067127	0.068767	0.439560
7	2097152	13.838519	0.033995	0.034771	3.736264
8	16777216	13.855745	0.017227	0.017545	30.109890
9	134217728	13.864453	0.008708	0.008837	240.769231

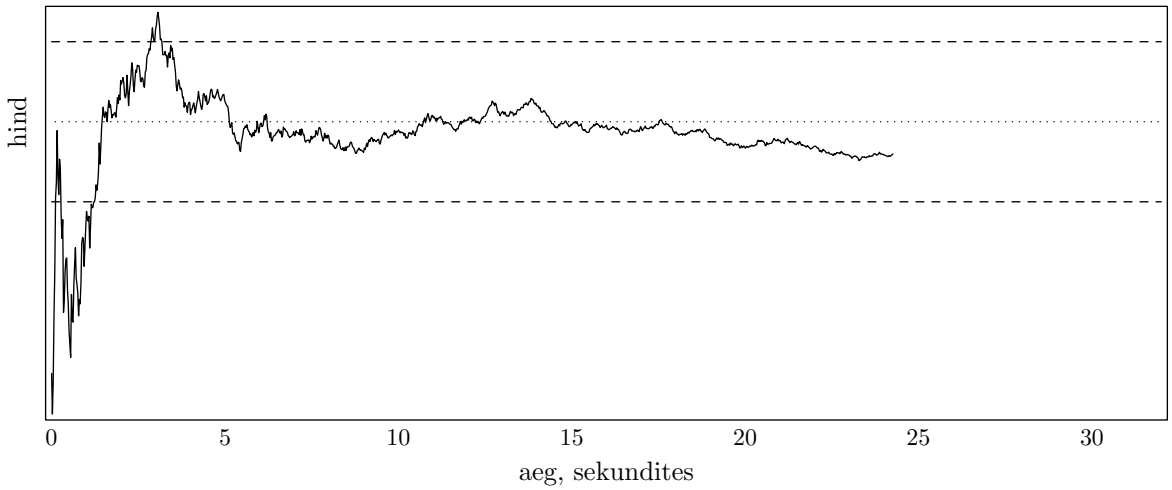
Tabel 5.10. Esimene võrestikumeetod kolmemõõtmelise ostuoptsiooni korral.

Samm	Punktide koguarv	Hinnalähend	Modifitseeritud Runge veahinnang	Tegelik viga	Aeg (s)
1	8	11.689747	5.844873	2.183543	0.000000
2	216	13.193812	0.752033	0.679478	0.000000
3	5832	13.634260	0.220224	0.239030	0.000000
4	157464	13.792005	0.078873	0.081285	0.219780
5	4251528	13.845732	0.026863	0.027558	6.978022
6	114791256	13.863986	0.009127	0.009304	188.351648

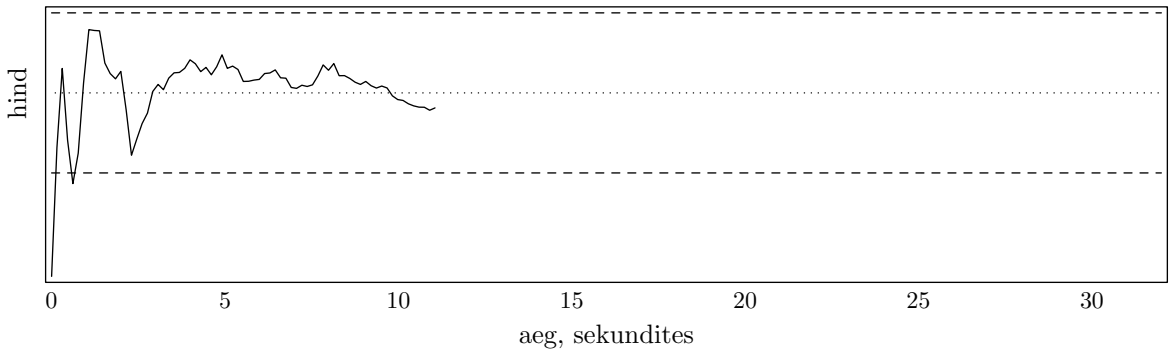
Tabel 5.11. Teine võrestikumeetod kolmemõõtmelise ostuoptsiooni korral.

Saab ligikaudselt hinnata, et järgmise sammu tegemine nõuaks kummaltki võrestikumeetodilt aega üle tunni, ja see on selge märk võrestikumeetodite peamisest puudusest.

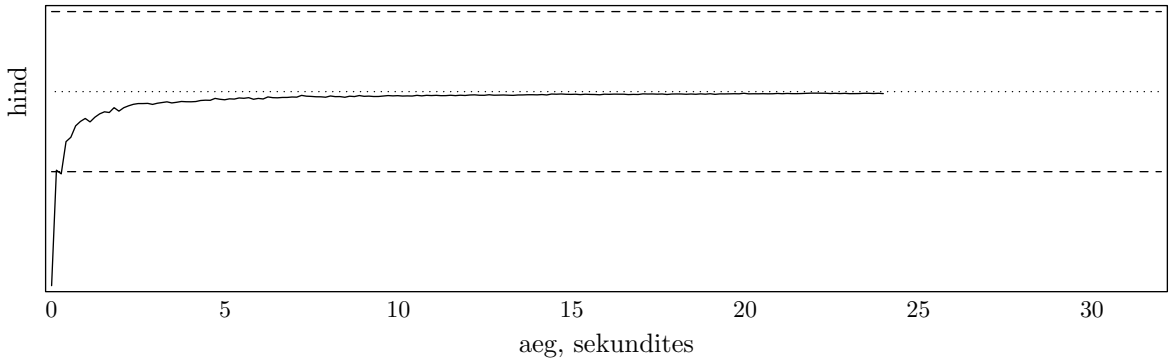
Hinnalähendite käitumist näitavad järgnevad joonised.



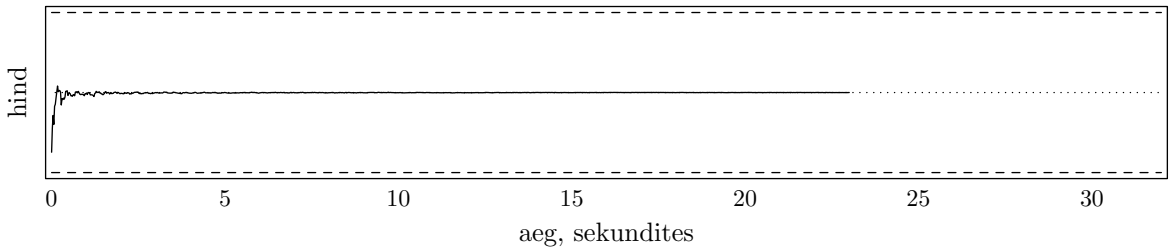
Joonis 5.29. Kolmemõõtmelise ostuoptsiooni hinnalähend Monte-Carlo meetodi korral.



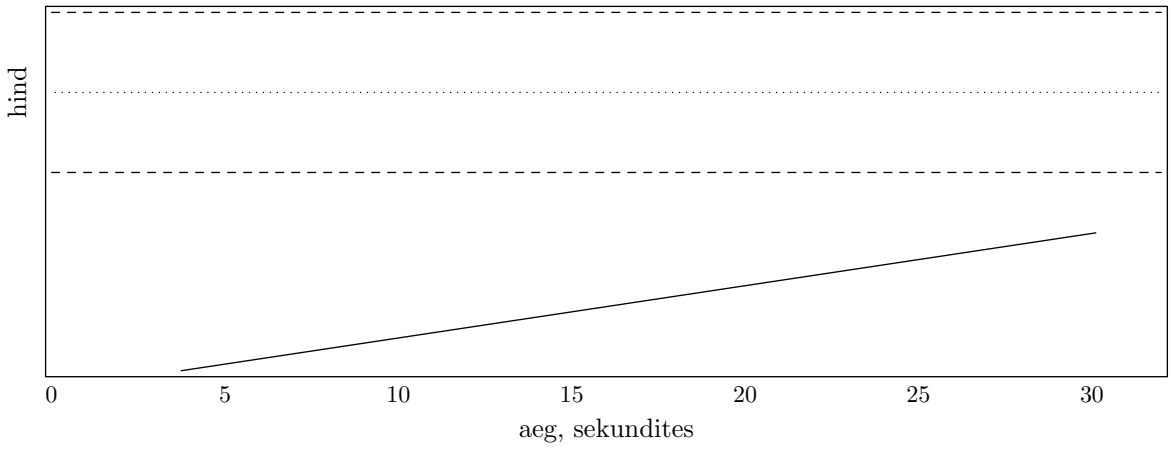
Joonis 5.30. Kolmemõõtmelise ostuoptsiooni hinnalähend antiteetilise Monte-Carlo meetodi korral.



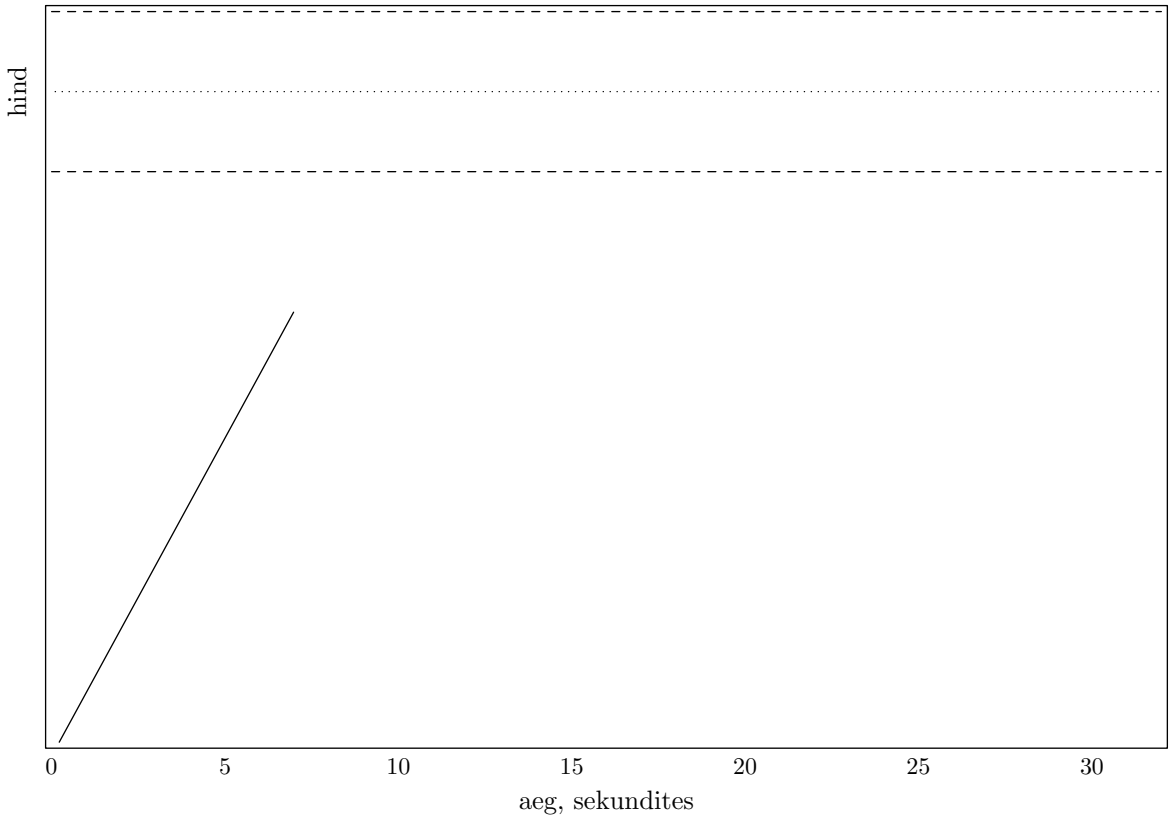
Joonis 5.31. Kolmemõõtmelise ostuoptsiooni hinnalähend kvaasi-Monte-Carlo meetodi korral (Haltoni jada).



Joonis 5.32. Kolmemõõtmelise ostuoptsiooni hinnalähend kvaasi-Monte-Carlo meetodi korral (Soboli jada).



Joonis 5.33. Kolmemõõtmelise ostuoptsiooni hinnalähend esimese võrestikumeetodi korral.

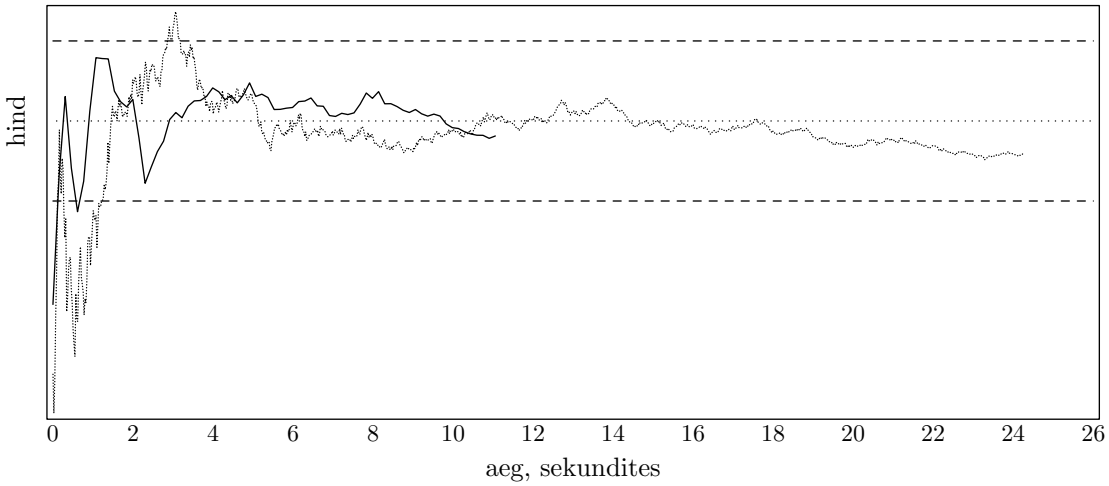


Joonis 5.34. Kolmemõõtmelise ostuoptsiooni hinnalähend teise võrestikumeetodi korral.

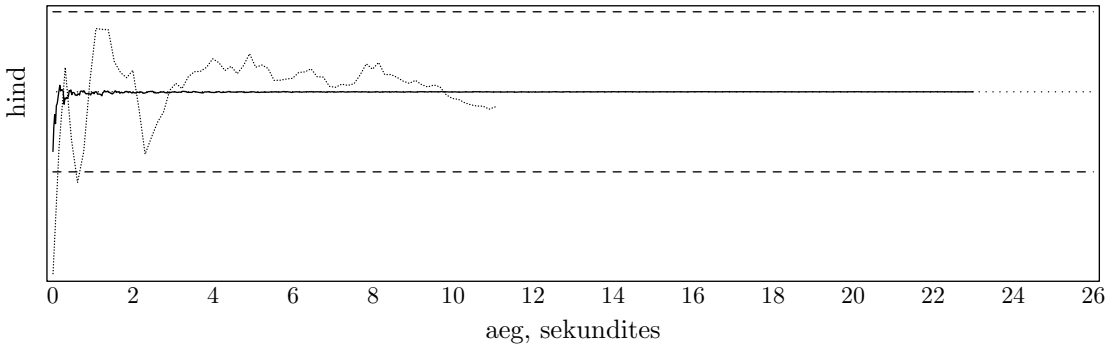
Saab teha järgmised järeldused:

- kumbki võrestikumeetod ei jõua anda piisavalt täpset vastust kahe minuti jooksul;
- teise võrestikumeetodi aeglustumine on esimese omast märgatavalt intensiivsem;
- Monte-Carlo meetod annab tulemuse rahuldava aja jooksul, kuid jääb siiski lootusetult maha mõlemast kvaasi-Monte-Carlo meetodist;
- antiteetilise Monte-Carlo meetodi kiirendus on väiksem kui kahemõõtmelisel juhul.

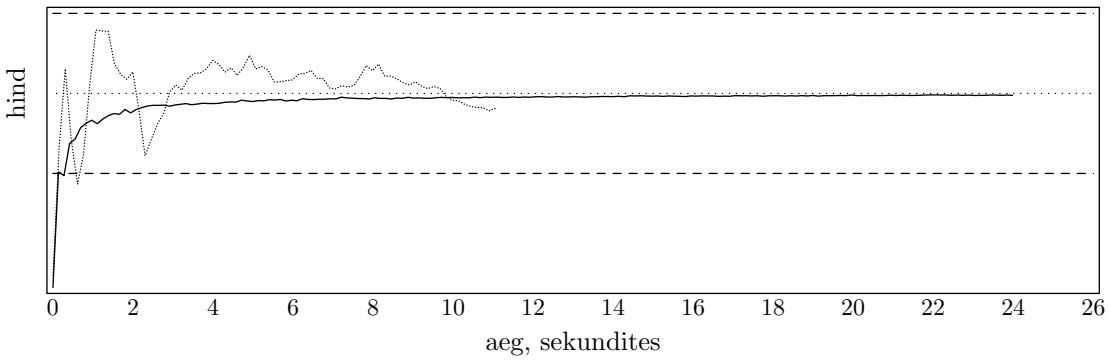
Seega hakkavad võrestikumeetodid mängust välja jääma, ning huvipakkuvate paaride hulk väheneb.



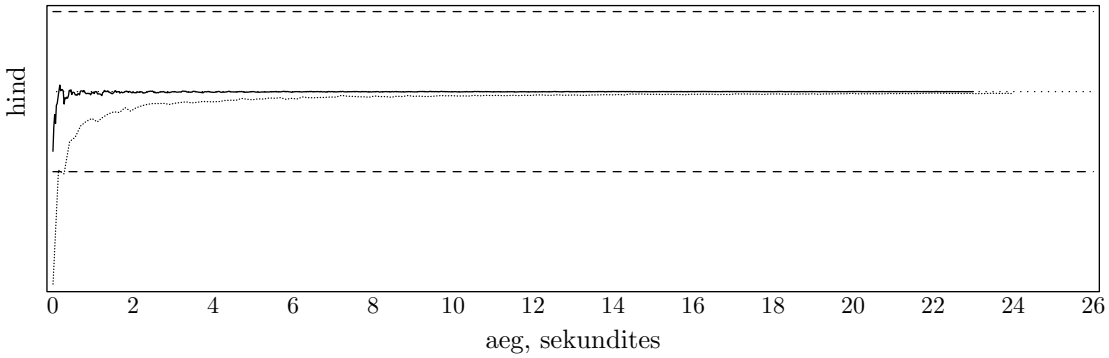
Joonis 5.35. Kolmemõõtmelise ostuoptsiooni hinnalähendid Monte-Carlo (punktjoon) ja antiteetilise Monte-Carlo meetodi korral.



Joonis 5.36. Kolmemõõtmelise ostuoptsiooni hinnalähendid antiteetilise Monte-Carlo (punktjoon) ja Soboli meetodi korral.



Joonis 5.37. Kolmemõõtmelise ostuoptsiooni hinnalähendid antiteetilise Monte-Carlo (punktjoon) ja Haltoni meetodi korral.



Joonis 5.38. Kolmemõõtmelise ostuoptsiooni hinnalähend Haltoni (punktjoon) ja Soboli meetodi korral.

Kvaasi-Monte-Carlo meetodid on nüüd teistest meetoditest selgelt kiiremad isegi jämedale veahinnangule vaatamata.

5.2.2. Müügioptsioon

Maksefunktsiooniks on kolmemõõtmelise Euroopa tüüpi müügioptsiooni maksefunktsioon:

$$p\big(S_1, S_2, S_3\big) = \max\{0, K_1 - S_1, K_2 - S_2, K_3 - S_3\}\,.$$

Täpne hind antud parameetrite korral on 9.17969. Eri meetodite tööajad on esitatud järgnevas tabelis.

Meetod	Aeg (s)
Monte-Carlo	8
Monte-Carlo, antiteetiline	1.5
kvaasi-Monte-Carlo, Haltoni jada	4
kvaasi-Monte-Carlo, Soboli jada	1
esimene võrestikumeetod	240
teine võrestikumeetod	190

Tabel 5.12. Eri meetodite ajad kolmemõõtmelise ostuoptsiooni hinna arvutamiseks.

Antiteetiline Monte-Carlo meetod on tavalisest Monte-Carlo meetodist umbes 5 korda kiirem, ning see on jällegi väiksem võit kahe aktsia juhuga võrreldes. Võrestikumeetodid arvutavad nii ostu- kui ka müügioptsiooni hinda sama ajaga, ja seda loomulikult oma iteratsioonipõhise ehituse tõttu (kuna nõutava täpsuse saamiseks kulub sama arv iteratsioone). Täpsemat infot iteratsioonide kohta annavad järgnevad tabelid.

Samm	Punktide koguarv	Hinnalähend	Modifitseeritud Runge veahinnang	Tegelik viga	Aeg (s)
1	8	8.216187	8.216187	0.963503	0.000000
2	64	8.763327	0.547139	0.416363	0.000000
3	512	8.953811	0.190485	0.225879	0.000000
4	4096	9.092267	0.138456	0.087423	0.000000
5	32768	9.137456	0.045188	0.042234	0.054945
6	262144	9.159530	0.022075	0.020160	0.439560
7	2097152	9.170121	0.010591	0.009569	3.736264
8	16777216	9.175203	0.005082	0.004487	30.219780
9	134217728	9.177588	0.002384	0.002102	243.351648

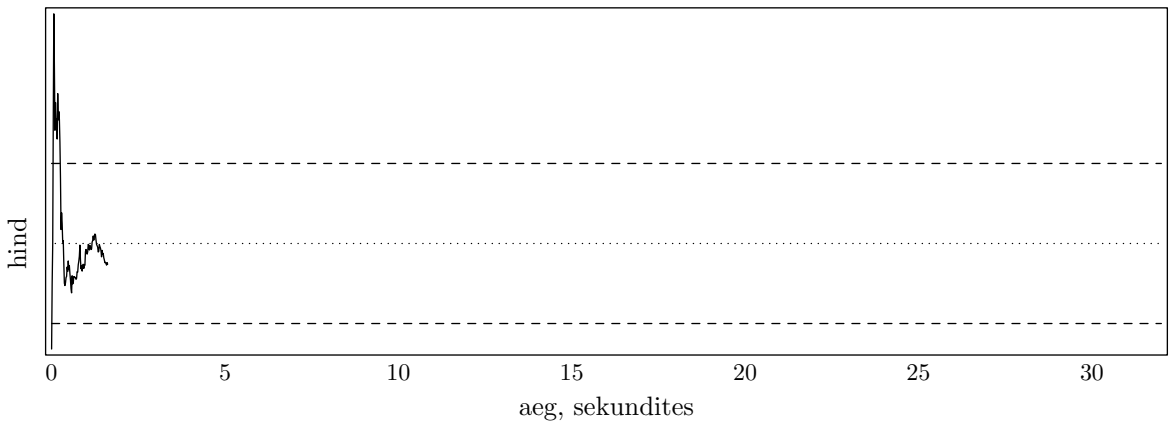
Tabel 5.13. Esimene võrestikumeetod kolmemõõtmelise müügioptsiooni korral.

Samm	Punktide koguarv	Hinnalähend	Modifitseeritud Runge veahinnang	Tegelik viga	Aeg (s)
1	8	8.216187	4.108094	0.963503	0.000000
2	216	8.950441	0.367127	0.229249	0.000000
3	5832	9.097052	0.073306	0.082638	0.000000
4	157464	9.155645	0.029297	0.024045	0.219780
5	4251528	9.172359	0.008357	0.007331	7.252747
6	114791256	9.177470	0.002555	0.002220	192.142857

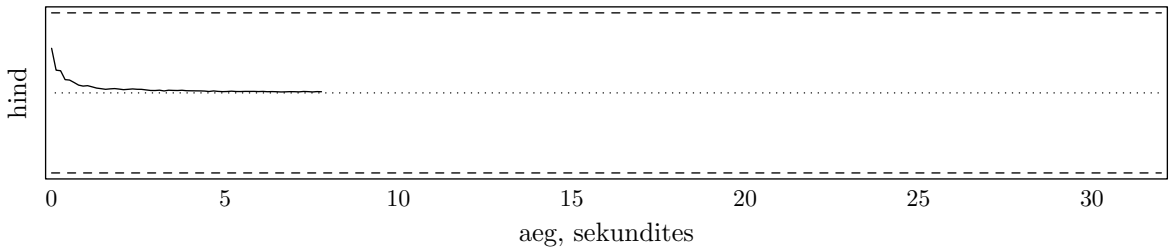
Tabel 5.14. Teine võrestikumeetod kolmemõõtmelise müügioptsiooni korral.

Hinnalähendite käitumist näitavad järgmised joonised.

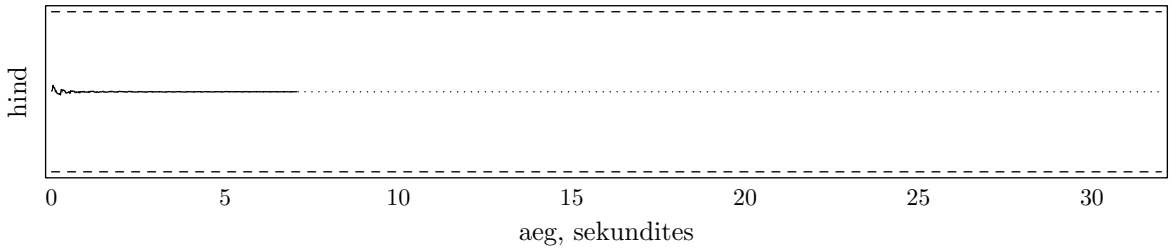
Joonis 5.39. Kolmemõõtmelise müügioptsiooni hinnalähend Monte-Carlo meetodi korral.



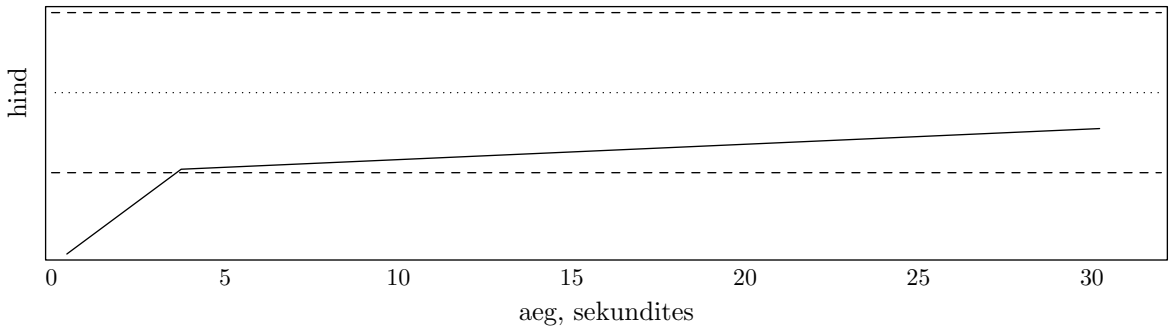
Joonis 5.40. Kolmemõõtmelise müügioptsiooni hinnalähend antiteetilise Monte-Carlo meetodi korral.



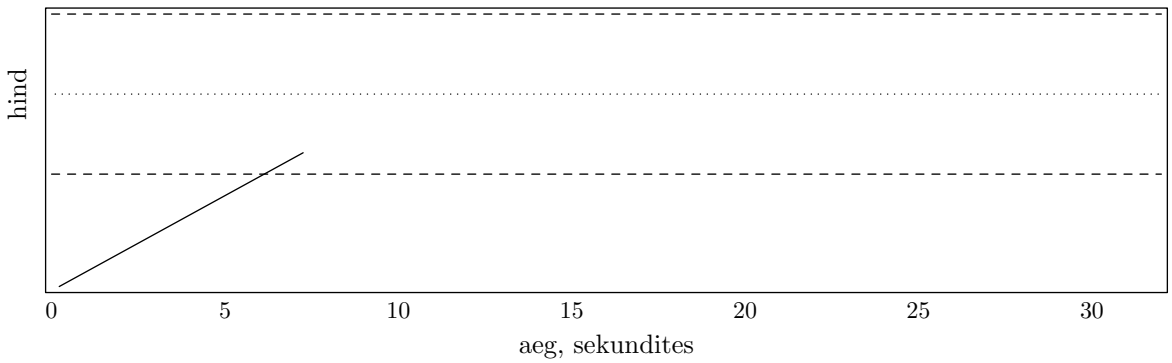
Joonis 5.41. Kolmemõõtmelise müügioptsiooni hinnalähend kvaasi-Monte-Carlo meetodi korral (Haltoni jada).



Joonis 5.42. Kolmemõõtmelise müügioptsiooni hinnalähend kvaasi-Monte-Carlo meetodi korral (Soboli jada).

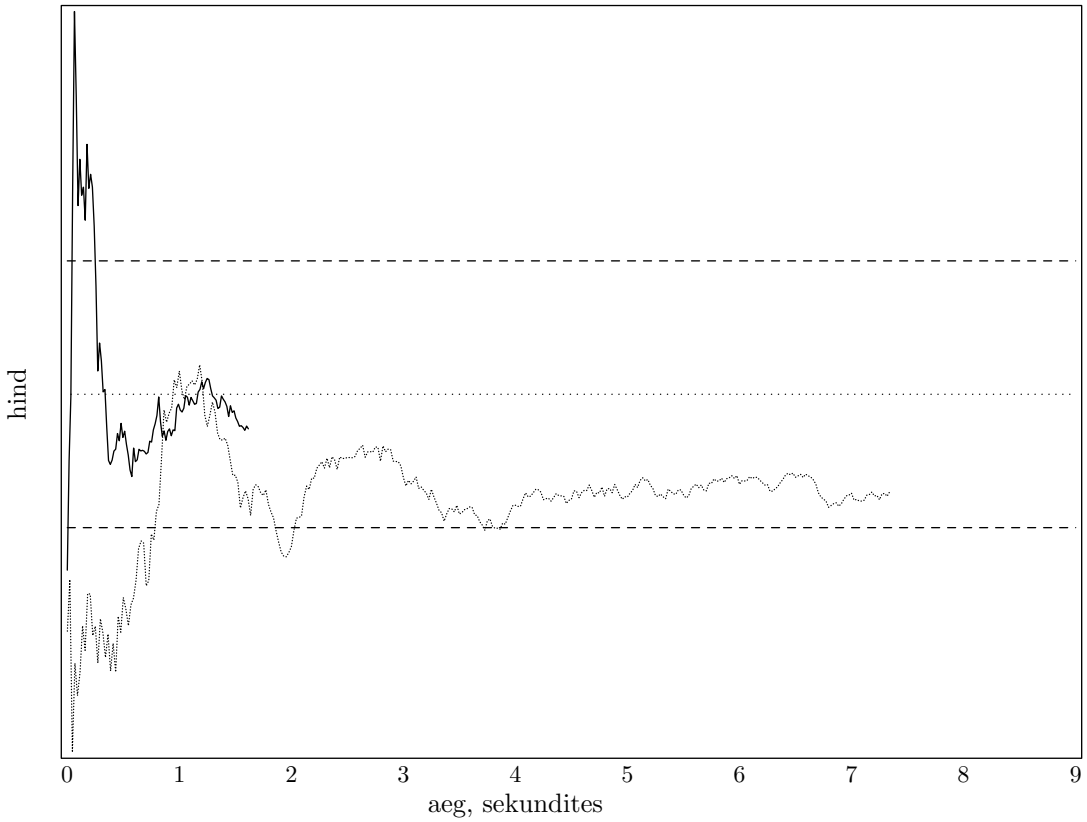


Joonis 5.43. Kolmemõõtmelise müügioptsiooni hinnalähend esimese võrestikumeetodi korral.

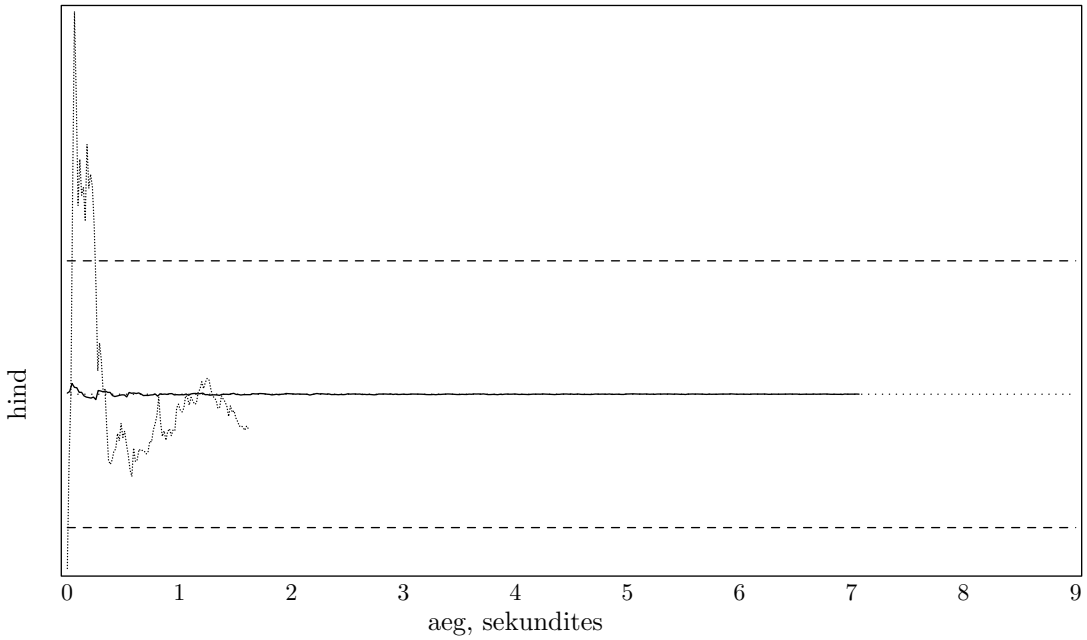


Joonis 5.44. Kolmemõõtmelise müügioptsiooni hinnalähend teise võrestikumeetodi korral.

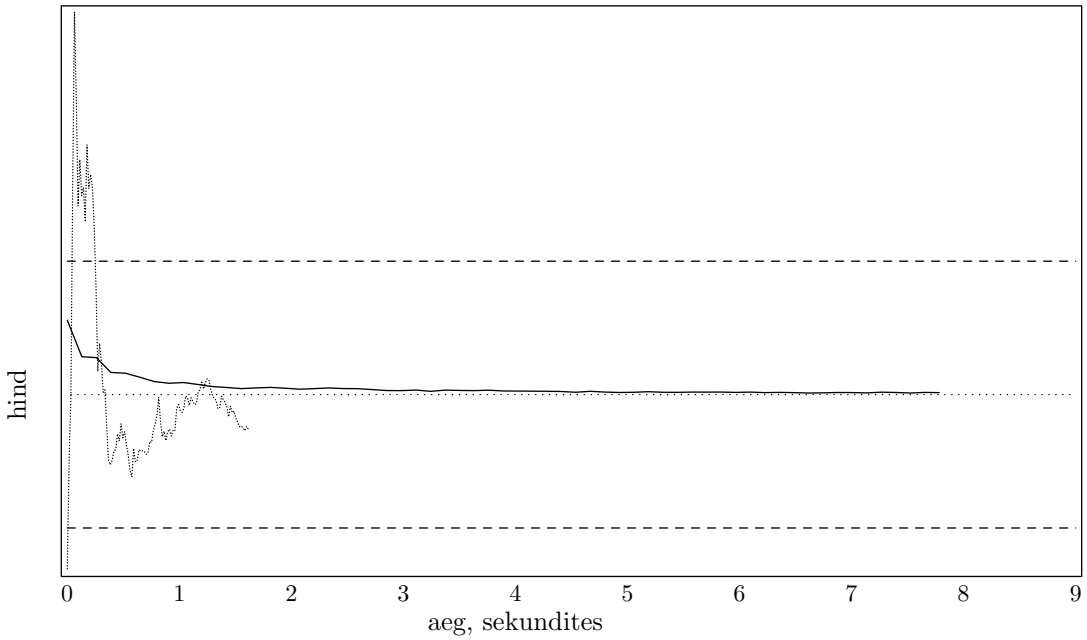
Märgime, et seekord võrestukumeetoditel õnnestus arvutada piisavalt täpne hind kahe minuti jooksul, kuid paarikaupa võrdlustel nad endiselt huvi ei paku.



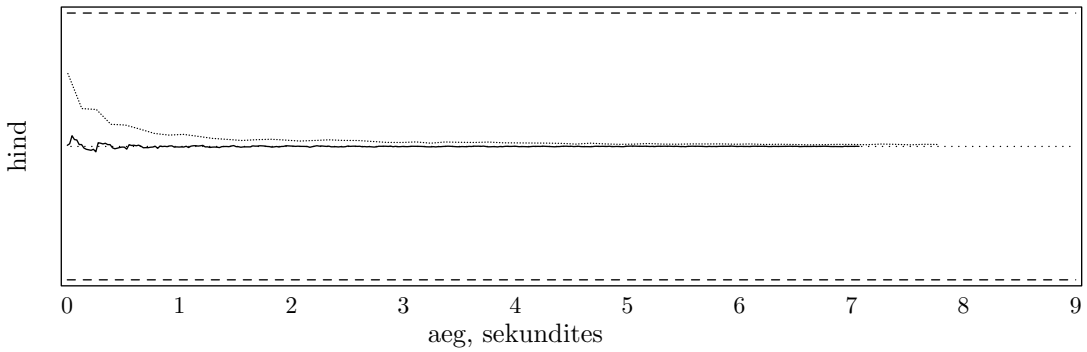
Joonis 5.45. Kolmemõõtmelise müügioptsiooni hinnalähendid Monte-Carlo (punktjoon) ja antiteetilise Monte-Carlo meetodi korral.



Joonis 5.46. Kolmemõõtmelise müügioptsiooni hinnalähendid antiteetilise Monte-Carlo (punktjoon) ja Soboli meetodi korral.



Joonis 5.47. Kolmemõõtmelise müügioptsiooni hinnalähendid antiteetilise Monte-Carlo (punktjoon) ja Haltoni meetodi korral.



Joonis 5.48. Kolmemõõtmelise müügioptsiooni hinnalähend Haltoni (punktjoon) ja Soboli meetodi korral.

Monte-Carlo meetodi seis on endiselt halb, võrestikumeetodite oma aga hoopis lootusetu.

5.3. Võrdluste kokkuvõte

Nüüd võtame kokku kogu läbitehtud eksperimendi ning teeme selle põhjal mõningaid järeldusi.

Mõlemad kvaasi-Monte-Carlo meetodid on näidanud ennast heast küljest. Vaadeldud näidetes edestavad nad teisi meetodeid. Haltoni algoritm on omakorda osutunud Soboli omast aeglasemaks.

Monte-Carlo meetod töötab aeglaselt, kuid siiski annab soovitud tulemuse mõistliku ajaga. Kiirendamine antiteetiliste juhuslike suuruste abil oluliselt parandab seda meetodit, kuid dimensiooni kasvades väheneb selle kiirendusmeetodi efektiivsus.

Võrestikumeetodid ehk mitmemõõtmelised kubatuurvalemid on üsna head kahemõõtmelisel juhul (ning ilmselt ka ühemõõtmelisel). Kolmemõõtmelisel juhul on nende kasutamise mõttekus juba küsitav, ning võib arvata, et juba neljamõõtmelisel juhul ei oma need meetodid praktilist tähtsust.

6. Kokkuvõte

Kvaasi-Monte-Carlo meetodit kasutatakse üha rohkem paljudes praktilistes ülesannetes Monte-Carlo meetodi asemel, sealhulgas ka integraalide väärtuste arvutamiseks.

Seetõttu sobivad kvaasi-Monte-Carlo meetodid optsioonide hindade arvutamiseks. Nagu ka Monte-Carlo meetodeid, saab neid hõlpsasti paralleliseerida.

Võrestikumeetodid, st. meetodid, mis kasutavad mitmemõõtmelisi kubatuurvalemeid, on alternatiiviks Monte-Carlo ja kvaasi-Monte-Carlo meetoditele. Võrestikumeetodid osutuvad väga headeks ühe- ja kahemõõtmelisel juhul, ka modifitseeritud Runge veahinnang töötab laitmatult. Kuid juhul, kui alusväärtpabereid on kolm või rohkem, osutuvad iteratsioonisammud liiga aeglasteks ning võrestikumeetodid jäävad teistele selgelt alla.

Kvaasi-Monte-Carlo meetodis võib kasutada erinevaid madala hälbimisega jadasid. Selles töös on vaadeldud Haltoni ja Soboli jadasid; selgeks soosikuks on osutunud Soboli algoritm. Mõlemad algoritmid on tunduvalt kiirema koondumisega kui Monte-Carlo meetod, ja seda vaatamata jämedale eksperthinnangule vea arvutamisel.

Monte-Carlo meetod — vana, hea ja lihtne meetod — on kvaasi-Monte-Carlo meetodite kõrval näidanud halvemat koondumisaega. Kuid ka kõrge dimensiooni korral töötab see meetod suhteliselt mõistliku ajaga; pealegi kiirendamine antiteetiliste arvude abil annab märgatava (2–10-kordse) võidu.

Kõigi mainitud meetodite korral on rohkesti võimalusi optimiseerimiseksja kiirendamiseks. Ning järjest rohkem kasutatakse optsioonide hindamisel uusi meetodeid, näiteks kiiret Fourier' teisendust või geneetilist programmeerimist[3].

Lõpetuseks võib öelda, et optsioonide hindamiseks kasutatavate meetodite hulk aina suureneb, neid meetodeid on suudetud ka mitmeti täiendada. Mõned eriti õnnestunud meetodid hakkavad asendama vanu ja aeglasemaid; ja nimelt kvaasi-Monte-Carlo meetodid on asendamas Monte-Carlo meetodeid.

[3] Geneetilise programmeerimise kohta on näiteks artikkel Chen, Shu-Heng; Yeh, Chia-Hsuan; Lee, Woh-Chiang. *Option Pricing with Genetic Programming.* Proceedings of 1997 International Conference of Genetic Algorithms. Morgan Kaufmann Publishers. Pp. 704–711.

О методах сеток, Монте-Карло и квази-Монте-Карло для вычисления стоимости опционов европейского типа

Резюме

Валерий Коорт

Цель данной работы — ознакомление с использованием методов Монте-Карло и квази-Монте-Карло, а также метода сеток для приближённого численного вычисления стоимости опционов — ценных бумаг, стоимость которых может зависеть от рыночной стоимости других ценных бумаг. Эта работа ограничивается рассмотрением опционов европейского типа.

Даётся представление о теоретической и алгоритмической сторонах методов, рассматривается вопрос оценки точности. На примере упрощённой модели рынка рассматриваются все стадии реализации соответствующих методов. К работе прилагаются тексты соответствующих программ. Проводится анализ некоторых из полученных численных результатов на предмет их согласованности с ожидаемыми.

Также проводится сравнение упомянутых методов между собой. Детально рассматривается шесть методов: Монте-Карло, Монте-Карло с использованием антитетических чисел, квази-Монте-Карло на основе алгоритма Халтона, квази-Монте-Карло на основе алгоритма Соболя и два различных метода сеток, являющихся в сущности многомерными кубатурными формулами.

Данная работая являтся продолжением и расширением бакалаврской работы автора (*О методе моделирования и методе биномиального дерева для вычисления стоимости опционов европейского типа*. Тартуский университет, 2001; на эстонском языке) и может быть использована как основа для более глубокого изучения описанных методов.

Kirjandus

[1] Acklam, Peter J. *An algorithm for computing the inverse normal cumulative distribution function.* `http://home.online.no/~pjacklam/notes/invnorm/`.

[2] Antonov, I.A.; Saleev, V.M. *An Economic Method of Computing LP_τ-sequences.* USSR Computational Mathematics and Mathematical Physics, 19, 252–256, 1979.

[3] Faure, Henry. *Monte-Carlo and Quasi-Monte-Carlo Methods for Numerical Integration.* Submitted for publication, 2001.

[4] Flannery, Brian P.; Press, William H.; Teukolsky, Saul A.; Vetterling, William T. *Numerical Recipes in C. The Art of Scientific Computing.* Second Edition. Cambridge University Press, 1992. `http://www.nr.com/`, `http://lib-www.lanl.gov/numerical/bookcpdf.html`.

[5] Halton, J.H. *On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals.* Numer. Math., 2, 84–90, 1960.

[6] Hickernell, Fred J.; Lemieux, Christiane; Owen, Art B. *Control variates for quasi-Monte Carlo.* Submitted for publication, 2002.

[7] Kollo, Tõnu. *Monte Carlo meetodid. Õpik.* Käsikiri. `http://www.ut.ee/~koortw/mcm/`.

[8] Koort, Valeri. *Simulatsiooni- ja puumeetoditest Eurooopa tüüpi optsioonide hindade arvutamisel. Bakalaureusetöö.* Tartu Ülikool, 2001.

[9] Niederreiter, Harald. *Random Number Generation and Quasi-Monte Carlo Methods.* Philadelphia, 1992.

[10] Papageorgiou, A.; Traub, J.F. *Faster Evaluation of Multidimensional Integrals.* Computational Physics, 11–6, 574–578, 1997.

[11] Paskov, Spassimir H. *New Metodologies for Valuing Derivatives.* Mathematics of Derivative Securities. Cambridge University Press, 1996.

[12] Sobol, I.M. *On the distribution of points in a cube and the approximate evaluation of integrals.* USSR Computational Mathematics and Mathematical Physics, 7, 86–112, 1967.

[13] Кнут, Дональд Эрвин. *Искусство программирования для ЭВМ. Том 2. Получисленные алгоритмы.* Москва, «Мир», 1977.

Lisad

A. Programm kahe aktsia jaoks

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>


// P~ohilised konstandid ning muutujad andmete vahetamiseks eri moodulite vahel
double eps = 0.01,           // Soovitav t"apsus
       alpha = 0.01,         // Eksimist~oen"aosus
       S0[2] = {40.0, 60.0}, // Alghinnad
       K[2] = {50.0, 50.0},  // Ostuhinnad
       r = 0.095310179804324860043952123280765 1, // Pangaprotsent, = ln(1.1)
       sigmaM[2][2] = {0.3, 0.0,                 // Volatiilsused,
                       0.0, 0.3},                //   maatrikskujul
       sigma[2], rho,        // Volatiilsused ja korrelatsioonikordaja,
                             //   alternatiivkujul, arvutatakse maatrikskujust
       a[2][2],              // Mitmem~o~otmelise normaaljaotuse genereerimiseks
       T = 0.5,              // T"aitmisaeg
       tapne = 13.47114540,  // T"apne hind - lihtsalt on teada
       x_vorestik[2],        // V~orestikumeetodi jaoks
       B0_ant[3][2],         // Antiteetilise Monte-Carlo jaoks
       S[2];                 // Modelleeritud aktsiahindade jaoks
long   mitu_kulus,           // Koondumiseks kulunud
                             //   aktsiahindade komplektide arv
       init;                 // randU() jaoks
char   meetod;               // N"aitab, mis meetodit kasutatakse
int    d = 2,                // QMC funktsioonide jaoks
       arve_varus = 0;       // Antiteetilise Monte-Carlo jaoks

/*****************************************************************************/
/*****************************************************************************/

void genereeri_hindu()
// Modelleerib aktsiahindade komplekti
{
    long i;
    double randU(), B[2], B0[2], x[2], halton_seq(), sobol_seq();
    long double Phi_inv(long double);

    if(meetod == 'H')
        halton_seq(&d, x);
    else if(meetod == 'S')
        sobol_seq(&d, x-1);
    else if(meetod == 'M')
        for(i=0; i<2; i++)
            x[i] = randU(&init);
    else if(meetod == 'A')
        ; // K~oik tehakse hiljem
    else if(meetod=='V' || meetod=='W')
        for(i=0; i<2; i++)
            x[i] = x_vorestik[i];
    else
        return;

    if(meetod == 'A')
```

```
56      {
57          if(arve_varus==0)
58          {
59              for(i=0; i<2; i++)
60              {
61                  x[i] = randU(&init);
62                  B0[i] = Phi_inv(x[i]);
63              }
64              B0_ant[0][0] = -B0[0]; B0_ant[0][1] = -B0[1];
65              B0_ant[1][0] = -B0[0]; B0_ant[1][1] =  B0[1];
66              B0_ant[2][0] =  B0[0]; B0_ant[2][1] = -B0[1];
67              arve_varus = 3;
68          }
69          else
70          {
71              for(i=0; i<2; i++)
72                  B0[i] = B0_ant[arve_varus-1][i];
73              arve_varus--;
74          }
75      }
76      else
77          for(i=0; i<2; i++)
78              B0[i] = Phi_inv(x[i]);
79
80      for(i=0; i<2; i++)
81          B[i] = a[i][0]*B0[0] + a[i][1]*B0[1];
82
83      for(i=0; i<2; i++)
84          S[i] = S0[i] * exp((r-sigma[i]*sigma[i]/2)*T + sigma[i]*B[i]);
85  }
86
87  /*****************************************************************************/
88
89  double p()
90  // Pseudojuhuslik maksefunktsiooni v"a"artus Euroopa ostuoptsiooni jaoks
91  {
92      double makse = 0.0, vahe;
93      long i;
94
95      genereeri_hindu();
96      for(i=0; i<2; i++)
97      {
98          vahe = S[i] - K[i];
99          // vahe = K[i] - S[i]; // m"u"ugioptsiooni jaoks
100         if(vahe > makse)
101             makse = vahe;
102     }
103     return makse;
104 }
105
106 /*****************************************************************************/
107
108 double f_N01(double x)
109 // Standardse normaaljaotuse N(0,1) tihedusfunktsioon
110 {
111     return  0.398942280401432 * exp(-x*x/2);
112     //      1/sqrt(2*Pi)
113 }
114
```

```
115 /*************************************************************************/
116
117 double Phi(double x)
118 // Standardse normaaljaotuse N(0,1) jaotusfunktsioon
119 {
120     double  k, k2, k3, k4, k5,
121     gamma = 0.2316419,  a1 =  0.319381530,  a2 = -0.356563782,
122     a3 = 1.781477937,   a4 = -1.821255978,  a5 =  1.330274429;
123
124     k = 1 + gamma*x;
125     k2 = k*k;
126     k3 = k2*k;
127     k4 = k3*k;
128     k5 = k4*k;
129
130     if(x < 0)
131         return 1-Phi(-x);
132     else
133         return 1 - f_N01(x)*(a1/k + a2/k2 + a3/k3 + a4/k4 + a5/k5);
134 }
135
136 /*************************************************************************/
137
138 double kvantiil(double alpha)
139 // N(0,1) alpha/2-t"aiendkvantiili leidmine kahendotsimisega
140 {
141     double k1 = -10.0, k2 = 10.0, k = 0.0, t = 1 - alpha/2;
142
143     while(fabs(Phi(k)-t) > 1e-10)     // t"apsusega 1e-10
144     {
145         if(Phi(k)-t > 0)
146             k2 = k;
147         else
148             k1=k;
149         k = (k1+k2) / 2;
150     }
151
152     return k;
153 }
154
155 /*************************************************************************/
156
157 double modelleeritud()
158 // Modelleerib optsioonihinda ja tagastab selle;
159 // salvestab, mitu pseudojuhuslikku makset selleks kulus
160 {
161     long n = 0;
162     FILE *f;
163     double h, disp = 1e80, sum, ruut_sum, uus_eps, kvantiil(double), modelleeritud_v();
164
165     if(meetod=='V' || meetod=='W') // Liiga erineva struktuuriga meetodid
166         return modelleeritud_v();
167
168     sum = ruut_sum = 0.0;
169
170     // Normaaljaotusega l"ahendamisel p~ohinev hinnang
171     uus_eps = eps/kvantiil(alpha);
172     uus_eps *= uus_eps;
173
```

```
174     printf(" punktide arv | l\"ahend | disp. hinnang | disp. t~oke | aeg\n");
175     while(disp > n*uus_eps || arve_varus>0)
176     {
177         h = p();
178         if(meetod == 'A')
179         {
180             while(arve_varus>0)
181                 h += p();
182             h /= 4;
183         }
184         sum += h;
185         ruut_sum += h*h;
186         // v"ahemalt 1000 makset tuleb genereerida - v"aljal"o"okide v"altimaks
187         if(++n > 1000)
188             disp = (ruut_sum - sum*sum/n) / (n - 1.0);
189         if(n%10000 == 0)
190         {
191             /*f = fopen("koondumine.txt", "a");
192             fprintf(f, "%f  %f\n",
193               ((double) clock())/CLOCKS_PER_SEC, exp(-r*T) * sum/n);
194             fclose(f);*/
195             if(n%100000 == 0)
196                 printf(" %ld: %f   %f %f   %lf sek\n",
197                        n, exp(-r*T) * sum/n, disp, n*uus_eps,
198                        ((double) clock())/CLOCKS_PER_SEC);
199         }
200     }
201
202     mitu_kulus = n;
203     return exp(-r*T) * sum/n;
204 }
205
206 /*************************************************************************/
207
208 double modelleeritud_v()
209 // Modelleerib optsioonihinda ja tagastab selle;
210 // kasutab esimest v~orestikumeetodit
211 {
212     long i, j, samm = 0, n=2, punkte;
213     FILE *f;
214     double sum = 0.0, In=0.0, Ikn, k=2.0;
215
216     if(meetod=='W')
217         k = 3.0;
218
219     printf(" punktide arv | l\"ahend | mod. Runge veahinnang | tegelik viga | aeg\n");
220     while(++samm < 15)
221     {
222         punkte = n * n;
223         if(meetod=='V')
224             sum = 0.0;
225         for(i = 0; i < n; i++)
226         {
227             x_vorestik[0] = (i+0.5)/n; // 1/2n + i*1/n
228             for(j = 0; j < n; j++)
229             {
230                 if(samm>1 && meetod=='W' && i%3==1 && j%3==1)
231                     continue; // eelmise sammu punktid
232                 x_vorestik[1] = (j+0.5)/n;
```

```
233                 sum += p();
234             }
235         }
236         In = Ikn;
237         Ikn = exp(-r*T) * sum/punkte;
238 
239         /*f = fopen("koondumine.txt", "a");
240         fprintf(f, "%f  %f\n",
241             ((double) clock())/CLOCKS_PER_SEC, exp(-r*T) * sum/punkte);
242         fclose(f);*/
243         printf(" %ld %f  %f  %f  %f sek\n",
244             punkte, Ikn, fabs(Ikn-In)/(k-1), fabs(Ikn-tapne),
245                 ((double) clock())/CLOCKS_PER_SEC);
246         if(meetod=='V')
247             n = 2*n;
248         if(meetod=='W')
249             n = 3*n;
250     }
251 
252     mitu_kulus = punkte;
253     return Ikn;
254 }
255 
256 /*****************************************************************************/
257 
258 // Standardse normaaljaotuse jaotusfunktsiooni p"o"ordfunktsioon.
259 // Allikad: http://home.online.no/~pjacklam/notes/invnorm/ ,
260 //          http://www.math.uio.no/~jacklam .
261 
262 #define  A_1  (-3.969683028665376e+01)
263 #define  A_2   2.209460984245205e+02
264 #define  A_3  (-2.759285104469687e+02)
265 #define  A_4   1.383577518672690e+02
266 #define  A_5  (-3.066479806614716e+01)
267 #define  A_6   2.506628277459239e+00
268 
269 #define  B_1  (-5.447609879822406e+01)
270 #define  B_2   1.615858368580409e+02
271 #define  B_3  (-1.556989798598866e+02)
272 #define  B_4   6.680131188771972e+01
273 #define  B_5  (-1.328068155288572e+01)
274 
275 #define  C_1  (-7.784894002430293e-03)
276 #define  C_2  (-3.223964580411365e-01)
277 #define  C_3  (-2.400758277161838e+00)
278 #define  C_4  (-2.549732539343734e+00)
279 #define  C_5   4.374664141464968e+00
280 #define  C_6   2.938163982698783e+00
281 
282 #define  D_1   7.784695709041462e-03
283 #define  D_2   3.224671290700398e-01
284 #define  D_3   2.445134137142996e+00
285 #define  D_4   3.754408661907416e+00
286 
287 #define P_LOW   0.02425
288 #define P_HIGH  0.97575
289 
290 long double Phi_inv(long double p)
291 {
```

```
292     long double x, q, r, u, e;
293     if ((0 < p )  && (p < P_LOW))
294     {
295         q = sqrt(-2*log(p));
296         x = (((((C_1*q+C_2)*q+C_3)*q+C_4)*q+C_5)*q+C_6) /
297                                             ((((D_1*q+D_2)*q+D_3)*q+D_4)*q+1);
298     }
299     else
300     {
301         if ((P_LOW <= p) && (p <= P_HIGH))
302         {
303             q = p - 0.5;
304             r = q*q;
305             x = (((((A_1*r+A_2)*r+A_3)*r+A_4)*r+A_5)*r+A_6)*q /
306                                     (((((B_1*r+B_2)*r+B_3)*r+B_4)*r+B_5)*r+1);
307         }
308         else
309         {
310             if ((P_HIGH < p) && (p < 1))
311             {
312                 q = sqrt(-2*log(1-p));
313                 x = -(((((C_1*q+C_2)*q+C_3)*q+C_4)*q+C_5)*q+C_6) /
314                                             ((((D_1*q+D_2)*q+D_3)*q+D_4)*q+1);
315             }
316         }
317     }
318     return x;
319 }
320
321 /*************************************************************************/
322
323 #define MAXDIM 6
324
325 void halton_seq(int *n, double x[])
326 // Haltoni jada genereeriv funktsioon, Soboli jada genereeriva funktsiooni
327 // sobol_seq() omakirjutatud analoog.
328 {
329     unsigned long a, j, kk, p, p_i, p_pow;
330     long i;
331     static unsigned long prime[MAXDIM] = {2, 3, 5, 7, 11, 13}, k;
332     double phi, logprime[MAXDIM]={0.69314718055994530941723212145818,
333         1.09861228866810969139524523692225,
334         1.60943791243410037460075933322262,
335         1.94591014905531330510535274344 32,
336         2.39789527279837054406194357796 51,
337         2.56494935746153673605348744156 53};
338
339     if (*n < 0)
340         k = 1;
341     else
342     {
343         for (j = 0; j < IMIN(*n,MAXDIM); j++)
344         {
345             p = prime[j];
346             p_pow = ceil(log(k)/logprime[j]);
347             phi = 0.0;
348             kk = k;
349             for(i = p_pow; i >= 0; i--)
350             {
```

```
                p_i = powi(p,i);
                a = kk / p_i;
                kk %= p_i;
                phi += a / ((double) p_i);
            }
            x[j] = phi / p;
        }
        k++;
    }
}

/*************************************************************************/

int IMIN(int a, int b)
{
    if (a<b) return a;
    else return b;
}

/*************************************************************************/

// J"argnevate funktsioonide allikaks on
// http://lib-www.lanl.gov/numerical/bookcpdf.html

/*************************************************************************/

#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

double randU(long *idum)
/* Long period (> 2 * 10^18) random number generator of L'Ecuyer
with Bays-Durham shuffle and added safeguards. Returns a uniform
random deviate between 0.0 and 1.0 (exclusive of the endpoint
values). Call with idum a negative integer to initialize; thereafter,
do not alter idum between successive deviates in a sequence. RNMX
should approximate the largest floating value that is less than 1.
*/
{
    int j;
    long k;
    static long idum2=123456789;
    static long iy=0;
    static long iv[NTAB];
    double temp;

    if (*idum <= 0) {
        if (-(*idum) < 1) *idum=1;
```

```
410         else *idum = -(*idum);
411         idum2=(*idum);
412         for (j=NTAB+7;j>=0;j--) {
413             k=(*idum)/IQ1;
414             *idum=IA1*(*idum-k*IQ1)-k*IR1;
415             if (*idum < 0) *idum += IM1;
416             if (j < NTAB) iv[j] = *idum;
417         }
418         iy=iv[0];
419     }
420     k=(*idum)/IQ1;
421     *idum=IA1*(*idum-k*IQ1)-k*IR1;
422     if (*idum < 0) *idum += IM1;
423     k=idum2/IQ2;
424     idum2=IA2*(idum2-k*IQ2)-k*IR2;
425     if (idum2 < 0) idum2 += IM2;
426     j=iy/NDIV;
427     iy=iv[j]-idum2;
428     iv[j] = *idum;
429     if (iy < 1) iy += IMM1;
430     if ((temp=AM*iy) > RNMX) return RNMX;
431     else return temp;
432 }
433
434 /*************************************************************************/
435
436 double randN(long *idum)
437 /* Returns a normally distributed deviate with zero mean and unit variance,
438 using randU(idum) as the source of uniform deviates. */
439 {
440     double randU(long *idum);
441     static int iset=0;
442     static double gset;
443     double fac,rsq,v1,v2;
444
445     if (*idum < 0) iset=0;
446     if (iset == 0) {
447         do {
448             v1=2.0*randU(idum)-1.0;
449             v2=2.0*randU(idum)-1.0;
450             rsq=v1*v1+v2*v2;
451         } while (rsq >= 1.0 || rsq == 0.0);
452         fac=sqrt(-2.0*log(rsq)/rsq);
453         gset=v1*fac;
454         iset=1;
455         return v2*fac;
456     } else {
457         iset=0;
458         return gset;
459     }
460 }
461
462 /*************************************************************************/
463
464 #define MAXBIT 30
465 #define MAXDIM 6
466
467 void sobol_seq(int *n, double x[])
468 /* When n is negative, internally initializes a set of MAXBIT direction
```

```
numbers for each of MAXDIM different Sobol' sequences. When n is positive
(but <=MAXDIM), returns as the vector x[1..n] the next values from n of
these sequences. (n must not be changed between initializations.) */
{
    int j,k,l;
    unsigned long i,im,ipp;
    static double fac;
    static unsigned long in,ix[MAXDIM+1],*iu[MAXBIT+1];
    static unsigned long mdeg[MAXDIM+1]={0,1,2,3,3,4,4};
    static unsigned long ip[MAXDIM+1]={0,0,1,1,2,1,4};
    static unsigned long iv[MAXDIM*MAXBIT+1]={
    0,1,1,1,1,1,1,3,1,3,3,1,1,5,7,7,3,3,5,15,11,5,15,13,9};
    if (*n < 0)
    {
        for (k=1;k<=MAXDIM;k++) ix[k]=0;
        in=0;
        if (iv[1] != 1) return;
        fac=1.0/(1L << MAXBIT);
        for (j=1,k=0;j<=MAXBIT;j++,k+=MAXDIM) iu[j] = &iv[k];
        for (k=1;k<=MAXDIM;k++)
        {
            for (j=1;j<=mdeg[k];j++) iu[j][k] <<= (MAXBIT-j);
            for (j=mdeg[k]+1;j<=MAXBIT;j++)
            {
                ipp=ip[k];
                i=iu[j-mdeg[k]][k];
                i ^= (i >> mdeg[k]);
                for (l=mdeg[k]-1;l>=1;l--)
                {
                    if (ipp & 1) i ^= iu[j-l][k];
                    ipp >>= 1;
                }
                iu[j][k]=i;
            }
        }
    }
    else
    {
        im=in++;
        for (j=1;j<=MAXBIT;j++)
        {
            if (!(im & 1)) break;
            im >>= 1;
        }
        if (j > MAXBIT) printf("MAXBIT too small in sobseq");
        im=(j-1)*MAXDIM;
        for (k=1;k<=IMIN(*n,MAXDIM);k++)
        {
            ix[k] ^= iv[im+k];
            x[k]=ix[k]*fac;
        }
    }
}

/*************************************************************************/
/*************************************************************************/

int main()
{
```

```
528     FILE *f;
529     int d_init = -1;
530     long i;
531     double aeg, x[2], m, sigmaB[2][2];
532     clock_t start;
533 
534     sobol_seq(&d_init, x-1);
535     halton_seq(&d_init, x);
536     init = -time(0);
537 
538     // Volatiilsuste ja korrelatsioonikordaja arvutamine volatiilsuste
539     // maatrikskujust l"ahtudes
540     for(i=0; i<2; i++)
541         sigma[i] = sqrt(sigmaM[i][0]*sigmaM[i][0] + sigmaM[i][1]*sigmaM[i][1]);
542     rho = (sigmaM[0][0]*sigmaM[1][0] + sigmaM[0][1]*sigmaM[1][1]) /
543                                                             (sigma[0]*sigma[1]);
544 
545     // Mitmem~o~otmelise normaaljaotuse genereerimiseks vajaliku maatriksi A leidmine
546     sigmaB[0][0] = sigmaB[1][1] = T;
547     sigmaB[0][1] = sigmaB[1][0] = rho * T;
548     a[0][1] = 0.0;
549     a[0][0] = sqrt(sigmaB[0][0]);
550     a[1][0] = sigmaB[1][0] / a[0][0];
551     a[1][1] = sqrt(sigmaB[1][1] - a[1][0]*a[1][0]);
552 
553     start = clock();
554 
555     meetod = 'M';
556     /*  'M' - Monte-Carlo
557         'A' - Monte-Carlo antiteetilistega
558         'H' - kvaasi-Monte-Carlo, Haltoni jada
559         'S' - kvaasi-Monte-Carlo, Soboli jada
560         'V' - esimene v~orestikumeetod
561         'W' - teine v~orestikumeetod
562     */
563 
564     m = modelleeritud();
565     aeg = ((double) clock() - start) / CLOCKS_PER_SEC;
566 
567     if(fabs(m-tapne) > eps)
568         i = 1;
569     else
570         i = 0;
571 
572     f = fopen("tulemused.txt", "a");
573     fprintf(f, "\n%d Ajaga %f, meetod=%c, n=%ld, hinnaks %f ",
574         i, aeg, meetod, mitu_kulus, m);
575     fclose(f);
576 
577     printf("\n\n%d Ajaga %f, meetod=%c, n=%ld, hinnaks %f ",
578         i, aeg, meetod, mitu_kulus, m);
579 
580     return 0;
581 }
```

B. Programm kolme aktsia jaoks

```
1 #include <stdio.h>
2 #include <stdlib.h>
```

```
3  #include <math.h>
4  #include <time.h>
5
6
7  // P~ohilised konstandid ning muutujad andmete vahetamiseks eri moodulite vahel
8  double eps = 0.01,           // Soovitav t"apsus
9         alpha = 0.01,         // Eksimist~oen"aosus
10        S0[3] = {40.0, 60.0, 50.0},              // Alghinnad
11        K[3] = {50.0, 50.0, 50.0},               // Ostuhinnad
12        r = 0.095310179804324860043952123280765 1, // Pangaprotsent, = ln(1.1)
13        sigmaM[3][3] = {0.3, 0.0, 0.1,           // Volatiilsused,
14                        0.0, 0.3, 0.1,           //   maatrikskujul
15                        0.1, 0.1, 0.2},
16        sigma[3],            // Volatiilsused ja korrelatsioonikordajad,
17        rho01, rho02, rho12, //   alternatiivkujul, arvutatakse maatrikskujust
18        a[3][3],             // Mitmem~o~otmelise normaaljaotuse genereerimiseks
19        T = 0.5,             // T"aitmisaeg
20        tapne = 13.87329,    // T"apne hind - lihtsalt on teada
21        x_vorestik[3],       // V~orestikumeetodi jaoks
22        B0_ant[15][3],       // Antiteetilise Monte-Carlo jaoks
23        S[3];                // Modelleeritud aktsiahindade jaoks
24 long   mitu_kulus,          // Koondumiseks kulunud
25                             //   aktsiahindade komplektide arv
26        init;                // randU() jaoks
27 char   meetod;              // N"aitab, mis meetodit kasutatakse
28 int    d = 3,               // QMC funktsioonide jaoks
29        arve_varus = 0;      // Antiteetilise Monte-Carlo jaoks
30
31 /*************************************************************************/
32 /*************************************************************************/
33
34 void genereeri_hindu()
35 // Modelleerib aktsiahindade komplekti
36 {
37     long i;
38     double randU(), B[3], B0[3], x[3], halton_seq(), sobol_seq();
39     long double Phi_inv(long double);
40
41     if(meetod == 'H')
42         halton_seq(&d, x);
43     else if(meetod == 'S')
44         sobol_seq(&d, x-1);
45     else if(meetod == 'M')
46         for(i=0; i<3; i++)
47             x[i] = randU(&init);
48     else if(meetod == 'A')
49         ; // K~oik tehakse hiljem
50     else if(meetod=='V' || meetod=='W')
51         for(i=0; i<3; i++)
52             x[i] = x_vorestik[i];
53     else
54         return;
55
56     if(meetod == 'A')
57     {
58         if(arve_varus==0)
59         {
60             for(i=0; i<3; i++)
61             {
```

```
62                  x[i] = randU(&init);
63                  B0[i] = Phi_inv(x[i]);
64              }
65              B0_ant[0][0] = -B0[0];  B0_ant[0][1] =  B0[1];  B0_ant[0][2] =  B0[2];
66              B0_ant[1][0] =  B0[0];  B0_ant[1][1] = -B0[1];  B0_ant[1][2] =  B0[2];
67              B0_ant[2][0] = -B0[0];  B0_ant[2][1] = -B0[1];  B0_ant[2][2] =  B0[2];
68              B0_ant[3][0] =  B0[0];  B0_ant[3][1] =  B0[1];  B0_ant[3][2] = -B0[2];
69              B0_ant[4][0] = -B0[0];  B0_ant[4][1] =  B0[1];  B0_ant[4][2] = -B0[2];
70              B0_ant[5][0] =  B0[0];  B0_ant[5][1] = -B0[1];  B0_ant[5][2] = -B0[2];
71              B0_ant[6][0] = -B0[0];  B0_ant[6][1] = -B0[1];  B0_ant[6][2] = -B0[2];
72              B0_ant[7][0] =  B0[0];  B0_ant[7][1] =  B0[2];  B0_ant[7][2] =  B0[1];
73              B0_ant[8][0] = -B0[0];  B0_ant[8][1] =  B0[2];  B0_ant[8][2] =  B0[1];
74              B0_ant[9][0] =  B0[0];  B0_ant[9][1] = -B0[2];  B0_ant[9][2] =  B0[1];
75              B0_ant[10][0] = -B0[0];  B0_ant[10][1] = -B0[2];  B0_ant[10][2] =  B0[1];
76              B0_ant[11][0] =  B0[0];  B0_ant[11][1] =  B0[2];  B0_ant[11][2] = -B0[1];
77              B0_ant[12][0] = -B0[0];  B0_ant[12][1] =  B0[2];  B0_ant[12][2] = -B0[1];
78              B0_ant[13][0] =  B0[0];  B0_ant[13][1] = -B0[2];  B0_ant[13][2] = -B0[1];
79              B0_ant[14][0] = -B0[0];  B0_ant[14][1] = -B0[2];  B0_ant[14][2] = -B0[1];
80              arve_varus = 15;
81          }
82          else
83          {
84              for(i=0; i<3; i++)
85                  B0[i] = B0_ant[arve_varus-1][i];
86              arve_varus--;
87          }
88      }
89      else
90          for(i=0; i<3; i++)
91              B0[i] = Phi_inv(x[i]);
92
93      for(i=0; i<3; i++)
94          B[i] = a[i][0]*B0[0] + a[i][1]*B0[1] + a[i][2]*B0[2];
95
96      for(i=0; i<3; i++)
97          S[i] = S0[i] * exp((r-sigma[i]*sigma[i]/2)*T + sigma[i]*B[i]);
98  }
99
100 /*************************************************************************/
101
102 double p()
103 // Pseudojuhuslik maksefunktsiooni v"a"artus Euroopa ostuoptsiooni jaoks
104 {
105     double makse = 0.0, vahe;
106     long i;
107
108     genereeri_hindu();
109     for(i=0; i<3; i++)
110     {
111         vahe = S[i] - K[i];
112         // vahe = K[i] - S[i]; // m"u"ugioptsiooni jaoks
113         if(vahe > makse)
114             makse = vahe;
115     }
116     return makse;
117 }
118
119 /*************************************************************************/
120
```

```
121 double f_N01(double x)
122 // Standardse normaaljaotuse N(0,1) tihedusfunktsioon
123 {
124     return  0.398942280401432 * exp(-x*x/2);
125     //      1/sqrt(2*Pi)
126 }
127
128 /*************************************************************************/
129
130 double Phi(double x)
131 // Standardse normaaljaotuse N(0,1) jaotusfunktsioon
132 {
133     double  k, k2, k3, k4, k5,
134     gamma = 0.2316419,  a1 =  0.319381530,  a2 = -0.356563782,
135     a3 = 1.781477937,   a4 = -1.821255978,  a5 =  1.330274429;
136
137     k = 1 + gamma*x;
138     k2 = k*k;
139     k3 = k2*k;
140     k4 = k3*k;
141     k5 = k4*k;
142
143     if(x < 0)
144         return 1-Phi(-x);
145     else
146         return 1 - f_N01(x)*(a1/k + a2/k2 + a3/k3 + a4/k4 + a5/k5);
147 }
148
149 /*************************************************************************/
150
151 double kvantiil(double alpha)
152 // N(0,1) alpha/2-t"aiendkvantiili leidmine kahendotsimisega
153 {
154     double k1 = -10.0, k2 = 10.0, k = 0.0, t = 1 - alpha/2;
155
156     while(fabs(Phi(k)-t) > 1e-10)     // t"apsusega 1e-10
157     {
158         if(Phi(k)-t > 0)
159             k2 = k;
160         else
161             k1=k;
162         k = (k1+k2) / 2;
163     }
164
165     return k;
166 }
167
168 /*************************************************************************/
169
170 double modelleeritud()
171 // Modelleerib optsioonihinda ja tagastab selle;
172 // salvestab, mitu pseudojuhuslikku makset selleks kulus
173 {
174     long n = 0;
175     FILE *f;
176     double h, disp=1e80, sum, ruut_sum, uus_eps, kvantiil(double), modelleeritud_v();
177
178     if(meetod=='V' || meetod=='W') // Liiga erineva struktuuriga meetodid
179         return modelleeritud_v();
```

```
180
181     sum = ruut_sum = 0.0;
182
183     // Normaaljaotusega l"ahendamisel p~ohinev hinnang
184     uus_eps = eps/kvantiil(alpha);
185     uus_eps *= uus_eps;
186
187     printf(" punktide arv | l\"ahend | disp. hinnang | disp. t~oke | aeg\n");
188     while(disp > n*uus_eps || arve_varus>0)
189     {
190         h = p();
191         if(meetod == 'A')
192         {
193             while(arve_varus>0)
194                 h += p();
195             h /= 16;
196         }
197         sum += h;
198         ruut_sum += h*h;
199         // v"ahemalt 1000 makset tuleb genereerida - v"aljal"o"okide v"altimaks
200         if(++n > 1000)
201             disp = (ruut_sum - sum*sum/n) / (n - 1.0);
202         if(n%10000 == 0)
203         {
204             /*f = fopen("koondumine.txt", "a");
205             fprintf(f, "%f  %f\n",
206               ((double) clock())/CLOCKS_PER_SEC, exp(-r*T) * sum/n);
207             fclose(f);*/
208             if(n%100000 == 0)
209                 printf(" %ld: %f   %f %f   %lf sek\n",
210                        n, exp(-r*T) * sum/n, disp, n*uus_eps,
211                        ((double) clock())/CLOCKS_PER_SEC);
212         }
213     }
214
215     mitu_kulus = n;
216     return exp(-r*T) * sum/n;
217 }
218
219 /*************************************************************************/
220
221 double modelleeritud_v()
222 // Modelleerib optsioonihinda ja tagastab selle;
223 // kasutab esimest v~orestikumeetodit
224 {
225     long i, j, h, samm = 0, n=2, punkte;
226     FILE *f;
227     double sum = 0.0, In=0.0, Ikn, k=2.0;
228
229     if(meetod=='W')
230         k = 3.0;
231
232     printf(" punktide arv | l\"ahend | mod. Runge veahinnang | tegelik viga | aeg\n");
233     while(++samm < 15)
234     {
235         punkte = n * n * n;
236         if(meetod=='V')
237             sum = 0.0;
238         for(i = 0; i < n; i++)
```

```
239         {
240             x_vorestik[0] = (i+0.5)/n; // 1/2n + i*1/n
241             for(j = 0; j < n; j++)
242             {
243                 x_vorestik[1] = (j+0.5)/n;
244                 for(h = 0; h < n; h++)
245                 {
246                     if(samm>1 && meetod=='W' && i%3==1 && j%3==1 && h%3==1)
247                         continue; // eelmise sammu punktid
248                     x_vorestik[2] = (h+0.5)/n;
249                     sum += p();
250                 }
251             }
252         }
253         In = Ikn;
254         Ikn = exp(-r*T) * sum/punkte;
255 
256         /*f = fopen("koondumine.txt", "a");
257         fprintf(f, "%f  %f\n",
258             ((double) clock())/CLOCKS_PER_SEC, exp(-r*T) * sum/punkte);
259         fclose(f);*/
260         printf(" %ld %f  %f  %f  %f sek\n",
261             punkte, Ikn, fabs(Ikn-In)/(k-1), fabs(Ikn-tapne),
262                 ((double) clock())/CLOCKS_PER_SEC);
263         if(meetod=='V')
264             n = 2*n;
265         if(meetod=='W')
266             n = 3*n;
267     }
268 
269     mitu_kulus = punkte;
270     return Ikn;
271 }
272 
273 /*************************************************************************/
274 
275 // Standardse normaaljaotuse jaotusfunktsiooni p"o"ordfunktsioon.
276 // Allikad: http://home.online.no/~pjacklam/notes/invnorm/ ,
277 //          http://www.math.uio.no/~jacklam .
278 
279 #define  A_1  (-3.969683028665376e+01)
280 #define  A_2   2.209460984245205e+02
281 #define  A_3  (-2.759285104469687e+02)
282 #define  A_4   1.383577518672690e+02
283 #define  A_5  (-3.066479806614716e+01)
284 #define  A_6   2.506628277459239e+00
285 
286 #define  B_1  (-5.447609879822406e+01)
287 #define  B_2   1.615858368580409e+02
288 #define  B_3  (-1.556989798598866e+02)
289 #define  B_4   6.680131188771972e+01
290 #define  B_5  (-1.328068155288572e+01)
291 
292 #define  C_1  (-7.784894002430293e-03)
293 #define  C_2  (-3.223964580411365e-01)
294 #define  C_3  (-2.400758277161838e+00)
295 #define  C_4  (-2.549732539343734e+00)
296 #define  C_5   4.374664141464968e+00
297 #define  C_6   2.938163982698783e+00
```

```
298
299 #define  D_1   7.784695709041462e-03
300 #define  D_2   3.224671290700398e-01
301 #define  D_3   2.445134137142996e+00
302 #define  D_4   3.754408661907416e+00
303
304 #define P_LOW   0.02425
305 #define P_HIGH  0.97575
306
307 long double Phi_inv(long double p)
308 {
309     long double x, q, r, u, e;
310     if ((0 < p )  && (p < P_LOW))
311     {
312         q = sqrt(-2*log(p));
313         x = (((((C_1*q+C_2)*q+C_3)*q+C_4)*q+C_5)*q+C_6) /
314                                         ((((D_1*q+D_2)*q+D_3)*q+D_4)*q+1);
315     }
316     else
317     {
318         if ((P_LOW <= p) && (p <= P_HIGH))
319         {
320             q = p - 0.5;
321             r = q*q;
322             x = (((((A_1*r+A_2)*r+A_3)*r+A_4)*r+A_5)*r+A_6)*q /
323                                 (((((B_1*r+B_2)*r+B_3)*r+B_4)*r+B_5)*r+1);
324         }
325         else
326         {
327             if ((P_HIGH < p) && (p < 1))
328             {
329                 q = sqrt(-2*log(1-p));
330                 x = -(((((C_1*q+C_2)*q+C_3)*q+C_4)*q+C_5)*q+C_6) /
331                                         ((((D_1*q+D_2)*q+D_3)*q+D_4)*q+1);
332             }
333         }
334     }
335     return x;
336 }
337
338 /*************************************************************************/
339
340 #define MAXDIM 6
341
342 void halton_seq(int *n, double x[])
343 // Haltoni jada genereeriv funktsioon, Soboli jada genereeriva funktsiooni
344 // sobol_seq() omakirjutatud analoog.
345 {
346     unsigned long a, j, kk, p, p_i, p_pow;
347     long i;
348     static unsigned long prime[MAXDIM] = {2, 3, 5, 7, 11, 13}, k;
349     double phi, logprime[MAXDIM]={0.69314718055994530941723212145818,
350         1.09861228866810969139524523692225,
351         1.60943791243410037460075933322620,
352         1.94591014905531330510535274344320,
353         2.39789527279837054406194357796510,
354         2.56494935746153673605348744156530};
355
356     if (*n < 0)
```

```
357         k = 1;
358     else
359     {
360         for (j = 0; j < IMIN(*n,MAXDIM); j++)
361         {
362             p = prime[j];
363             p_pow = ceil(log(k)/logprime[j]);
364             phi = 0.0;
365             kk = k;
366             for(i = p_pow; i >= 0; i--)
367             {
368                 p_i = powi(p,i);
369                 a = kk / p_i;
370                 kk %= p_i;
371                 phi += a / ((double) p_i);
372             }
373             x[j] = phi / p;
374         }
375         k++;
376     }
377 }
378
379 /*********************************************************************/
380
381 int IMIN(int a, int b)
382 {
383     if (a<b) return a;
384     else return b;
385 }
386
387 /*********************************************************************/
388
389 // J"argnevate funktsioonide allikaks on
390 // http://lib-www.lanl.gov/numerical/bookcpdf.html
391
392 /*********************************************************************/
393
394 #define IM1 2147483563
395 #define IM2 2147483399
396 #define AM (1.0/IM1)
397 #define IMM1 (IM1-1)
398 #define IA1 40014
399 #define IA2 40692
400 #define IQ1 53668
401 #define IQ2 52774
402 #define IR1 12211
403 #define IR2 3791
404 #define NTAB 32
405 #define NDIV (1+IMM1/NTAB)
406 #define EPS 1.2e-7
407 #define RNMX (1.0-EPS)
408
409 double randU(long *idum)
410 /* Long period (> 2 * 10^18) random number generator of L'Ecuyer
411 with Bays-Durham shuffle and added safeguards. Returns a uniform
412 random deviate between 0.0 and 1.0 (exclusive of the endpoint
413 values). Call with idum a negative integer to initialize; thereafter,
414 do not alter idum between successive deviates in a sequence. RNMX
415 should approximate the largest floating value that is less than 1.
```

```
*/
{
    int j;
    long k;
    static long idum2=123456789;
    static long iy=0;
    static long iv[NTAB];
    double temp;

    if (*idum <= 0) {
        if (-(*idum) < 1) *idum=1;
        else *idum = -(*idum);
        idum2=(*idum);
        for (j=NTAB+7;j>=0;j--) {
            k=(*idum)/IQ1;
            *idum=IA1*(*idum-k*IQ1)-k*IR1;
            if (*idum < 0) *idum += IM1;
            if (j < NTAB) iv[j] = *idum;
        }
        iy=iv[0];
    }
    k=(*idum)/IQ1;
    *idum=IA1*(*idum-k*IQ1)-k*IR1;
    if (*idum < 0) *idum += IM1;
    k=idum2/IQ2;
    idum2=IA2*(idum2-k*IQ2)-k*IR2;
    if (idum2 < 0) idum2 += IM2;
    j=iy/NDIV;
    iy=iv[j]-idum2;
    iv[j] = *idum;
    if (iy < 1) iy += IMM1;
    if ((temp=AM*iy) > RNMX) return RNMX;
    else return temp;
}

/*************************************************************************/

double randN(long *idum)
/* Returns a normally distributed deviate with zero mean and unit variance,
using randU(idum) as the source of uniform deviates. */
{
    double randU(long *idum);
    static int iset=0;
    static double gset;
    double fac,rsq,v1,v2;

    if (*idum < 0) iset=0;
    if (iset == 0) {
        do {
            v1=2.0*randU(idum)-1.0;
            v2=2.0*randU(idum)-1.0;
            rsq=v1*v1+v2*v2;
        } while (rsq >= 1.0 || rsq == 0.0);
        fac=sqrt(-2.0*log(rsq)/rsq);
        gset=v1*fac;
        iset=1;
        return v2*fac;
    } else {
        iset=0;
```

```
475         return gset;
476     }
477 }
478 
479 /*************************************************************************/
480 
481 #define MAXBIT 30
482 #define MAXDIM 6
483 
484 void sobol_seq(int *n, double x[])
485 /* When n is negative, internally initializes a set of MAXBIT direction
486 numbers for each of MAXDIM different Sobol' sequences. When n is positive
487 (but <=MAXDIM), returns as the vector x[1..n] the next values from n of
488 these sequences. (n must not be changed between initializations.) */
489 {
490     int j,k,l;
491     unsigned long i,im,ipp;
492     static double fac;
493     static unsigned long in,ix[MAXDIM+1],*iu[MAXBIT+1];
494     static unsigned long mdeg[MAXDIM+1]={0,1,2,3,3,4,4};
495     static unsigned long ip[MAXDIM+1]={0,0,1,1,2,1,4};
496     static unsigned long iv[MAXDIM*MAXBIT+1]={
497     0,1,1,1,1,1,1,3,1,3,3,1,1,5,7,7,3,3,5,15,11,5,15,13,9};
498     if (*n < 0)
499     {
500         for (k=1;k<=MAXDIM;k++) ix[k]=0;
501         in=0;
502         if (iv[1] != 1) return;
503         fac=1.0/(1L << MAXBIT);
504         for (j=1,k=0;j<=MAXBIT;j++,k+=MAXDIM) iu[j] = &iv[k];
505         for (k=1;k<=MAXDIM;k++)
506         {
507             for (j=1;j<=mdeg[k];j++) iu[j][k] <<= (MAXBIT-j);
508             for (j=mdeg[k]+1;j<=MAXBIT;j++)
509             {
510                 ipp=ip[k];
511                 i=iu[j-mdeg[k]][k];
512                 i ^= (i >> mdeg[k]);
513                 for (l=mdeg[k]-1;l>=1;l--)
514                 {
515                     if (ipp & 1) i ^= iu[j-l][k];
516                     ipp >>= 1;
517                 }
518                 iu[j][k]=i;
519             }
520         }
521     }
522     else
523     {
524         im=in++;
525         for (j=1;j<=MAXBIT;j++)
526         {
527             if (!(im & 1)) break;
528             im >>= 1;
529         }
530         if (j > MAXBIT) printf("MAXBIT too small in sobseq");
531         im=(j-1)*MAXDIM;
532         for (k=1;k<=IMIN(*n,MAXDIM);k++)
533         {
```

```
534             ix[k] ^= iv[im+k];
535             x[k]=ix[k]*fac;
536         }
537     }
538 }
539
540 /*************************************************************************/
541 /*************************************************************************/
542
543 int main()
544 {
545     FILE *f;
546     int d_init = -1;
547     long i;
548     double aeg, x[3], m, sigmaB[3][3];
549     clock_t start;
550
551     sobol_seq(&d_init, x-1);
552     halton_seq(&d_init, x);
553     init = -time(0);
554
555     // Volatiilsuste ja korrelatsioonikordaja arvutamine volatiilsuste
556     // maatrikskujust l"ahtudes
557     for(i=0; i<3; i++)
558         sigma[i] = sqrt(sigmaM[i][0]*sigmaM[i][0] + sigmaM[i][1]*sigmaM[i][1]
559                         + sigmaM[i][2]*sigmaM[i][2]);
560     rho01 = (sigmaM[0][0]*sigmaM[1][0] + sigmaM[0][1]*sigmaM[1][1] +
561              sigmaM[0][2]*sigmaM[1][2]) / (sigma[0]*sigma[1]);
562     rho02 = (sigmaM[0][0]*sigmaM[2][0] + sigmaM[0][1]*sigmaM[2][1] +
563              sigmaM[0][2]*sigmaM[2][2]) / (sigma[0]*sigma[2]);
564     rho12 = (sigmaM[2][0]*sigmaM[1][0] + sigmaM[2][1]*sigmaM[1][1] +
565              sigmaM[2][2]*sigmaM[1][2]) / (sigma[2]*sigma[1]);
566
567     // Mitmem~o~otmelise normaaljaotuse genereerimiseks vajaliku maatriksi A leidmine
568     sigmaB[0][0] = sigmaB[1][1] = sigmaB[2][2] = T;
569     sigmaB[0][1] = sigmaB[1][0] = rho01 * T;
570     sigmaB[0][2] = sigmaB[2][0] = rho02 * T;
571     sigmaB[2][1] = sigmaB[1][2] = rho12 * T;
572     a[0][1] = a[0][2] = a[1][2] = 0.0;
573     a[0][0] = sqrt(sigmaB[0][0]);
574     a[1][0] = sigmaB[1][0] / a[0][0];
575     a[1][1] = sqrt(sigmaB[1][1] - a[1][0]*a[1][0]);
576     a[2][0] = sigmaB[0][2] / a[0][0];
577     a[2][1] = (sigmaB[1][2] - a[1][0]*a[2][0]) / a[1][1];
578     a[2][2] = sqrt(sigmaB[2][2] - a[2][0]*a[2][0] - a[2][1]*a[2][1]);
579
580     start = clock();
581
582     meetod = 'M';
583     /*  'M' - Monte-Carlo
584         'A' - Monte-Carlo antiteetilistega
585         'H' - kvaasi-Monte-Carlo, Haltoni jada
586         'S' - kvaasi-Monte-Carlo, Soboli jada
587         'V' - esimene v~orestikumeetod
588         'W' - teine v~orestikumeetod
589     */
590
591     m = modelleeritud();
592     aeg = ((double) clock() - start) / CLOCKS_PER_SEC;
```

```
593
594     if(fabs(m-tapne) > eps)
595         i = 1;
596     else
597         i = 0;
598
599     f = fopen("tulemused.txt", "a");
600     fprintf(f, "\n%d Ajaga %f, meetod=%c, n=%ld, hinnaks %f ",
601         i, aeg, meetod, mitu_kulus, m);
602     fclose(f);
603
604     printf("\n\n%d Ajaga %f, meetod=%c, n=%ld, hinnaks %f ",
605         i, aeg, meetod, mitu_kulus, m);
606
607     return 0;
608 }
```

C. Kompakt-disk

Käesolevale tööle on lisatud üks laserplaat, mille peal on töö koopia PDF-formaadis, programmide lähtetekstid ning ka valik abiprogramme, sh. kompilaator ja tarkvara PDF-failide vaatamiseks.

- Juurdirektooriumis on töö tekst PDF-failina ning kahe programmi lähtetekstid (kahe ja kolme aktsia jaoks).
- Kaustas Tagavarakoopiad asuvad kolme ülalnimetatud faili koopiad. Koopiaid on nii originaalkujul kui ka ZIP- ja RAR-arhiividena; kokku on 8 koopiate komplekti.
- Kaustas Abitarkvara on valik programme, mis võivad osutuda vajalikuks antud tööga tutvumisel.
  - Acrobat Reader — vabatarkvara PDF-failide vaatamiseks. On esitatud neli erinevat versiooni: 3.0, 4.05, 5.1 ning 6.0. Installeerimisel võiks lähtuda põhimõttest, et vanem versioon nõuab vähem ruumi ja tavaliselt on rahul ka aeglasema arvutiga, kuid seetõttu sisaldab ka vähem võimalusi failiga opereerimiseks.
  - djgpp — käsurea kompilaator (djgpp 2.03, vabavara), mida on kasutatud programmide jooksutamiseks.
  - libseq beta 04.21.01 — teegid madala hälbimusega jadade genereerimiseks. Lähtetekstid; arendusstaadiumis olev vabavara. Käesolevas töös ei ole leidnud kasutust realisatsiooni erinevuste tõttu (libseq on keeles C++ ning objektorienteeritud struktuuriga), kuid võib olla kasulik antud temaatikast huvitatule.
  - WinRAR ning UnRAR — vastavalt jaosvara ning vabavara RAR-arhiviide käsitlemiseks. WinRAR võimaldab ka arhiveerimist ning ZIP-arhiivide käsitlemist.