VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
FUNDAMENTINIŲ MOKSLŲ FAKULTETAS
MATEMATINĖS STATISTIKOS KATEDRA

Paulius Bistrickis

GAUSO PROCESŲ MODELIAVIMAS IR ALGORITMŲ PALYGINIMAS

SIMULATION OF GAUSSIAN PROCESSES AND COMPARISON OF ALGORITHMS

Baigiamasis magistro darbas

Taikomosios statistikos studijų programa, valstybinis kodas 621G31001
Statistinių metodų finansuose ir ekonomikoje specializacija
Statistikos studijų kryptis

Vilnius, 2014

Vilniaus Gedimino technikos universitetas
Fundamentinių mokslų fakultetas
Matematinės statistikos katedra

ISBN ISSN
Egz. sk.
Data · · · · · · – · · · – · · ·

Taikomosios statistikos studijų programos baigiamasis magistro darbas
Pavadinimas **Gauso procesų modeliavimas ir algoritmų palyginimas**
Autorius **Paulius Bistrickis** Vadovas prof. habil. dr. **Kęstutis Kubilius**

Kalba
x lietuvių
anglų

Anotacija
Darbas yra apie trupmeninio Brauno proceso generavimo algoritmus. Tai yra vienas populiariausių Gauso procesų. Tikslas yra palyginti keletą dažniausiai naudojamų generavimo algoritmų ir išrinkti geriausią. Algoritmai lyginami spartos, gausiškumo bei tikslumo aspektais. Didelė dalis skiriama Hursto parametro įvertinimo metodams, išrenkamas algoritmas, kuris tai atlieka tiksliausiai. Vėliau geriausias metodas naudojamas generavimo algoritmų tikslumui nustatyti. Skaičiavimai ir generavimas atliktas naudojant R paketą, priede pateikiami generavimo ir vertinimo algoritmų programiniai kodai.

Prasminiai žodžiai: Gauso procesas, Hursto parametro vertinimo algoritmai, trupmeninis Brauno procesas, trupmeninio Brauno proceso generavimo algoritmai.

Vilniaus Gediminas technical university
Fundamental sciences faculty
Mathematical statistics department

ISBN ISSN
Copies. No.
Date · · · · · · – · · · – · · ·

Applied statistics study programme master thesis
Title **Simulation of Gaussian Processes and Comparison of Algorithms**
Author **Paulius Bistrickis** Executive prof. habil. dr. **Kęstutis Kubilius**

Thesis language
x Lithuanian
Foreign (English)

Abstract
This paper is about simulation methods of the fractional Brownian motion, which is one of the most popular Gaussian process. The main idea was to compare simulation methods and to find the best one. Criteria were speed, gaussianity and accuracy. Also, there are some information about the Hurst parameter estimation methods. It was found the most accurate estimation method. All calculation and simulation were made by using R project. In appendices it can be found source code.

Key words: fractional Brownian motion, Gaussian processes, Hurst parameter estimation, simulation of the fractional Brownian motion.

Turinys

1. Įvadas

Tiriant įvairius reiškinius, neapsieinama be atsitiktininių procesų. Vienas žinomiausių yra Gauso procesas. Jis yra įdomus tiek savo teorine prasme, tiek praktiniu pritaikymu. Gauso procesas svarbus matematinei statistikai, ekonometriniam modeliavimui, biologijai, inžinerijai. Kad būtų galima geriau pažinti procesą, ištirti jo savybes, elgseną, reikėtų mokėti jį modeliuoti. Tam yra sukurta labai daug algorimtų ir tikriausiai visi jie modeliuoja teisingai, tačiau priklausomai nuo algoritmo vieni reikalauja daugiau skaičiavimų, kiti mažiau. Dauguma algoritmų remiasi matricų skaičiavimais, kurios kartais būna milžiniškos, dirbti su tokiomis matricomis yra sudėtinga, reikia daug laiko ir resursų. Dėl šių priežasčių yra svarbu rasti geriausius algoritmus, kurie optimaliausiai generuotų reikalingų parametrų atsitiktinių procesų realizacijas. Kadangi Gauso procesas yra labai platus, šiame darbe apsiribosime atskiru Gauso proceso atveju — trupmeniniu Brauno judesiu. Kita vertus, mokant generuoti trupmeninį Brauno judesį, algoritmą nesunkiai galima pritaikyti ir generuojant paprastą Gauso procesą.

Šio darbo tikslas — palyginti trupmeninio Brauno judesio modeliavimo algoritmus ir rasti geriausią. Darbo uždaviniai yra:

- modeliuoti trupmeninį Brauno judesį naudojant įvairius algoritmus;
- remiantis kriterijais palyginti algoritmus ir išrinkti geriausią.

Pirmiausia apžvelgsime literatūrą, kas jau yra padaryta pasaulyje ir Lietuvoje kalbant apie trupmeninį Brauno judesį. Prisiminsime Gauso proceso, trupmeninio Brauno judesio apibrėžimus, savybes. Toliau pereisime prie algoritmų aprašymo, modeliavimo. Atrinksime vertinimo metodus, kurie leistų palyginti algoritmus tarpusavyje. Modeliavimą ir generavimą atliksime naudodami $\mathbb{R}$ paketą.

2. Literatūros apžvalga

Yra sukurta daug algoritmų, skirtų trupmeniniam Brauno judesiui generuoti. Vieni algoritmai jį generuoja tiesiogiai, kiti skirti generuoti Gauso procesui, o tada naudojantis tuo, kad Brauno judesys yra kumuliacinis Gauso procesas, perskaičiuojant gaunamas Brauno judesys. Norint generuoti Brauno judesį iš esmės pakanka rasti proceso kovariacinės matricos išskaidymą į kitų matricų sandaugą. Dauguma sukurtų algoritmų ir yra sukurti bandant kuo efektyviau tai padaryti. Pats tiksliausias algoritmas, bet kartu ir neefektyviausias yra Cholesky dekompozicija. Kadangi naudojant Cholesky algoritmą tenka skaičiuoti atvirkštines matricas, metodas yra lėtas, ypač generuojant daug atsitiktinių dydžių. Kita grupė algoritmų yra asimptotiniai. Jie yra žymiai greitesni, bet nėra tokie tikslūs. Daug vilčių dedama į vadinamus spektrinius algoritmus [4]. Tiriant juos, išsiaiškinta, kad jie yra gana tikslūs, tačiau žymiai greitesni nei kiti algoritmai. Greičiausiu ir tiksliausiu metodu laikomas ciklinės matricos metodas, dar vadinamas Wood–Chano[6].

Trupmeninio Brauno judesio populiarumo pradžia laikoma 1968 m., kai jį savo darbe aprašė Mandelbrotas ir Van Nessas [9]. Savo darbe remsimės keliais straipsniais, kuriuose nagrinėjami trupmeninio Brauno judesio generavimo algoritmai. Vienas jų yra „Simulation and indentification of the fractional Brownian motion: a bibliographical and comparative study“parašytas Jean–François Coeurjolly 2000 m., kuriame nagrinėjami kai kurie generavimo metodai, pateikiami keli lyginimo testai, bei realizuoti algoritmai naudojant S–plus programą. Kitas straipsnis, kuriuo remsimės, yra „On spectral simulation of fractional Brownian motion“, parašytas 2003 m. autorių Antonius B. Dieker ir Michel Mandjes. Šiame darbe dėmesys teikiamas asimptotiniams ir spektriniams metodams, juos laikant potencialiai efektyviausiais algoritmais, kurie leidžia generuoti trupmeninį Brauno judesį. 2004 m. tas pats Antonius B. Dieker papildė šį straipsnį ir parašė naują darbą „Simulation of fractional Brownian motion“, kuriame galima rasti dar daugiau algoritmų ir testų.

Didelis dėmesys skiriamas ir Hursto parametro įvertinimui. Kuriami vis tobulesni metodai kaip tai atlikti tiksliau ir greičiau. Savo darbe pamėginsime realizuoti metodus aprašytus Bretono ir Coeurjolly 2011 m. darbe [2]. Autoriai pateikia iš pirmo žvilgsnio patrauklų metodą H parametrui vertinti, pasikliovimo intervalams sudaryti, gaunami neįtikėtini rezultatai, kuriuos mėginsime patikrinti. Kitas naujas darbas, kuriuo remsimės yra Salomono ir Forto (2013 m.), čia aprašomas tiek trupmeninio Brauno judesio generavimo, tiek H vertinimo metodas [11]. Norėdami palyginti, kuris algoritmas veikia geriau, pasirinkome kelis Coeurjolly [6] bei keletą D.Melichovo [10] nagrinėtų ir realizuotų vertinimo metodų.

3. Apibrėžimai

Gauso procesui apibrėžti yra svarbi Gauso atsitiktinio dydžio sąvoka. Tai toks atsitiktinis dydis, kuris turi normalųjį skirstinį.

1 apibrėžimas. *Sakome, kad atsitiktinis dydis X turi normalųjį (Gauso) skirstinį su vidurkiu μ ir dispersija σ^2, jei tas atsitiktinis dydis įgyja reikšmes iš intervalo $(-\infty;\infty)$, o jo tankio funkcija turi pavidalą:*

$$f_{(x|\mu,\sigma)} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{\sigma^2}}, \tag{1}$$

čia $\sigma > 0$, o $-\infty < \mu < \infty$. Žymima $X \sim \mathcal{N}(\mu, \sigma^2)$. X vadinamas standartiniu normaliuoju, kai $X \sim \mathcal{N}(0,1)$

2 apibrėžimas. *n–matis atsitiktinis vektorius $\mathbb{X} = (X_1, X_2 ... X_n)^T$ yra normalusis tada ir tik tada, kai kiekvienam n–mačiui vektoriui $a = (a_1, a_2, ... a_n)^T$, vienmatis atsitiktinis dydis $a^T\mathbb{X}$ yra normalusis.*

3 apibrėžimas. *Atsitiktinis procesas yra atsitiktinių dydžių rinkinys $\{X_t, t \in \mathbb{T}\}$, čia $\mathbb{T} \subset \mathbb{R}$ yra indeksų aibė.*

Gauso procesą galima apibrėžti, kaip atsitiktinį procesą, sudarytą iš Gauso atsitiktinių dydžių.

4 apibrėžimas. *Procesas $\{X_t, t \in \mathbb{T}\}$ yra vadinamas Gauso, jei jo baigtiniamačiai skirstiniai yra normalieji (Gauso). Skirstinio savybės nusakomos vidurkio funkcija $\mu(t) = \mathbb{E}X_t$ ir kovariacine funkcija $\Gamma(t,s) = \mathbb{E}X_tX_s$.*

5 apibrėžimas. *Tolydus centruotas Gauso procesas $\{B_t^H, t \in \mathbb{T}\}$, $B_0^H = 0$, kurio kovariacinė funkcija turi pavidalą:*

$$\Gamma(t,s) = \mathbb{E}\left(B_t^H B_s^H\right) = \frac{1}{2}\left(t^{2H} + s^{2H} - |t-s|^{2H}\right), \forall t, s \geqslant 0 \tag{2}$$

yra vadinamas trupmeniniu Brauno judesiu (žym. tBj). $H \in (0;1)$ vadinamas Hursto parametru.

Kai Hursto parametras lygus $\frac{1}{2}$, procesas yra standartinis Brauno judesys. Trupmeninio Brauno judesio skirtumai yra vadinami trupmeniniu Gauso procesu (žym. tGp).

6 apibrėžimas. *Trupmeniniu Brauno judesiu vadinamas atsitiktinis procesas* $\{B_t^H, t \in \mathbb{T}\}$, $B_0^H = 0$, *turintis tokią stochastinę išraišką:*

$$B_t^H := \frac{1}{\Gamma(H+\frac{1}{2})}\left(\int_{-\infty}^{0}[(t-s)^{H-\frac{1}{2}}-(-s)^{H-\frac{1}{2}}]dB_s+\int_0^t(t-s)^{H-\frac{1}{2}}dB_s\right), \quad (3)$$

čia $\Gamma(\alpha) := \int_{-\infty}^{0} x^{\alpha-1}\exp(-x)dx$, $H \in (0;1)$ *yra Hursto parametras, o* B_s *yra standartinis Brauno judesys (su* $H = 1/2$*).*

Trupmeninis Brauno judesys ir trupmeninis Gauso procesas yra tarpusavyje susiję.

7 apibrėžimas. *Trupmeninis Gauso judesio procesas* $\{X_k, k \in \mathbb{N} \cup \{0\}\}$ *yra apibrėžiamas kaip tBj proceso skirtumas*

$$X_k = B_{k+1}^H - B_k^H, \quad (4)$$

turintis autokovariacinę funkciją:

$$\gamma(k) = \mathbb{E}X_{n+k}X_n = \frac{1}{2}\left(|k-1|^{2H} - 2|k|^{2H} + |k+1|^{2H}\right). \quad (5)$$

8 apibrėžimas. *p eilės filtras a yra apibrėžiamas* $l+1$ *ilgio vektoriumi, kurio komponentės* a_i *yra realios,* $0 \leqslant i \leqslant l$*, ir kuris tenkina:*

$$\sum_{q=0}^{l} q^j a_q = 0, \quad j = 0, ..., p-1 \quad ir \quad \sum_{q=0}^{l} q^p a_q \neq 0. \quad (6)$$

4. Modeliavimo algoritmai

Toliau apžvelgsime algoritmus, skirtus trupmeniniam Brauno judesiui modeliuoti. Algoritmai gali būti skirstomi į tiesioginius ir netiesioginius. Tiesioginiais algoritmais vadintume tokius metodus, kurie remiasi stochastinio integralo aproksimavimu, kovariacinės matricos išdėstymu. Netiesioginiai metodai pagrįsti trupmeninio Gauso proceso kovariacinės matricos savybėmis, kai generuojamas tGp, o iš jo gaunamas tBj. Iš tiesų, jei sugebame generuoti tBj realizaciją, apskaičiavę skirtumus, gauname trupmeninį Gauso procesą. Atvirkščiai, jei sugebame generuoti trupmeninį Gauso procesą, sumuodami galime gauti trupmeninį Brauno judesį.

4.1. Stochastinio integralo aproksimavimas

Šis algoritmas remiasi trupmeninio Brauno judesio stochastinės išraiškos aproksimavimu. Iš apibrėžimo, tBj galima išreikšti per stochastinių integralų sumą:

$$B_t^H := \frac{1}{\Gamma(H+\frac{1}{2})}\left(\int_{-\infty}^{0}[(t-s)^{H-\frac{1}{2}}-(-s)^{H-\frac{1}{2}}]dB_s+\int_0^t(t-s)^{H-\frac{1}{2}}dB_s\right). \quad (7)$$

Aproksimuojant šie integralai išreiškiami per sumas:

$$\tilde{B}_n^H := C_H\left(\sum_{k=-b}^{0}[(n-k)^{H-\frac{1}{2}}-(-k)^{H-\frac{1}{2}}]B_{1,k}+\sum_{k=0}^{n}(n-k)^{H-\frac{1}{2}}B_{2,k}\right), \quad (8)$$

čia B_1 yra nepriklausomų vienodai pasiskirsčiusių standartinių normaliųjų atsitiktinių dydžių $b+1$ ilgio vektorius, o B_2 yra nepriklausomų vienodai pasiskirsčiusių standartinių normaliųjų atsitiktinių dydžių $N+1$ ilgio vektorius. Sugeneravę tokią sumą naudodami B_1 ir B_2, gauname trupmeninį Brauno judesį. Literatūroje [6] rašoma, kad šis algoritmas vertingesnis savo istorine prasme nei praktine ir kad tai nėra geras būdas generuoti tBj.

4.2. Tikrinių reikšmių metodas

Šis metodas paremtas kovariacinės matricos Γ

$$\Gamma(t,s)=\mathbb{E}\left(B_t^H B_s^H\right)=\frac{1}{2}\left(t^{2H}+s^{2H}-|t-s|^{2H}\right), \forall t,s \geqslant 0 \quad (9)$$

išdėstymu, naudojant tikrines reikšmes. Kadangi Γ yra simetrinė, teigiamai apibrėžta matrica, tai jos tikrinės reikšmės $\lambda_i \geqslant 0$ $(i=1,...,N)$. Tegul $\Lambda_{i,j}=\lambda_i\delta_{i,j}$ yra diagonalioji tikrinių reikšmių matrica, čia $\delta_{i,j}$ yra Kronekerio simbolis. Pažymime matricą $\Lambda_{i,j}^{\frac{1}{2}}=\lambda_i^{\frac{1}{2}}\delta_{i,j}$. Tegul v_i yra tikrinis vektorius, susietas su tikrine reikšme λ_i. Apibrėžiame matricą P, kaip matricą, kurios i–tasis stulpelis yra tikrinis vektorius v_i. Kadangi tikriniai vektoriai yra tiesiškai nepriklausomi, egzistuoja atvirkštinė matrica P^{-1}. Tada egzistuoja tokia matrica $\Sigma = P\Lambda^{\frac{1}{2}}P^{-1}$, kad $\Sigma^2=\Gamma$. Padauginę Σ iš nepriklausomų standartinių normaliųjų atsitiktinių dydžių vektoriaus gauname trupmeninį Brauno judesį.

Šio algoritmo realizavimą $\mathbb{R}$ paketu galima rasti priede.

4.3. Cholesky metodas

Šis algoritmas remiasi trupmeninio Brauno judesio kovariacijų matricos Cholesky dekompozicija. Tarkime $\Gamma = (\frac{i}{N}; \frac{j}{N})$, čia $i, j \in \{0, ..., N-1\}$ yra tBj kovariacijų matrica. Iš kovariacijų matricos pašalinę pirmą stulpelį ir pirmą eilutę sudarome naują matricą, kurią pažymime $\grave{\Gamma} = (\frac{i}{N}; \frac{j}{N})$, tik čia $i, j \in \{1, ..., N-1\}$. Kadangi $\grave{\Gamma}$ yra teigiamai apibrėžta simetrinė matrica, tai ji turi vadinamąją Cholesky dekompoziciją $\grave{\Gamma} = LL'$, čia L yra trikampės matricos. Tokią trikampę matricą L padauginę iš nepriklausomų standartinių normaliųjų atsitiktinių dydžių vektoriaus gauname trupmeninį Brauno judesį.

Nors algoritmas yra laikomas tiksliu, bet jis pagrįstas matricų daugyba, o tokie skaičiavimai reikalauja daug resursų. Vis dėlto rašant šį darbą, pavyko patobulinti algoritmą pašalinus buvusius ciklus ir taip pagreitinus jį apie 10 kartų. Priede pateikiame šio algoritmo, realizuoto $\mathbb{R}$ paketu, programinį kodą.

4.4. Karhunen – Loeve generavimo metodas

Šio algoritmo idėja yra Gauso proceso išreiškimas per Karhunen – Loeve (toliau — K–L) išdėstymą. Pirmiausia naudojant parametrą H generuojama kovariacijų matrica, turinti pavidalą:

$$\Gamma(t,s) = \mathbb{E}\left(B_t^H B_s^H\right) = \frac{1}{2}\left(t^{2H} + s^{2H} - |t-s|^{2H}\right), \qquad \forall t, s \geqslant 0. \quad (10)$$

Baigtiniam n KL išdėstymas atrodo taip:

$$B_{t_i}^H \approx \sum_{k \geqslant 1}^{n} \sqrt{\hat{\lambda}_k}\xi_k \hat{\varphi}_{k,t_i}, \qquad \forall i = 0, 1, ..., n, \quad (11)$$

čia $\hat{\lambda_k}$ yra kovariacijų matricos tikrinės reikšmės, $\hat{\varphi_k}$ tikriniai vektoriai, o ξ_k yra nepriklausomų vienodai pasiskirsčiusių normaliųjų atsitiktinių dydžių seka. Kadangi sekos generavimui šiuo algoritmu reikia kovariacinių matricų aprašymo bei tikrinių reikšmių radimo, algoritmas nėra labai greitas. Savo struktūra algoritmas panašus į anksčiau aprašytus Cholesky bei tikrinių reikšmių metodus. Daugiau informacijos apie šį algoritmą galima rasti Luis A. Salomon straipsnyje [11], o prieduose $\mathbb{R}$ paketu realizuoto algoritmo programinį kodą.

4.5. Levinsono algoritmas

Norint generuoti trupmeninį Brauno judesį galima tai atlikti netiesiogiai. Levinsono algoritmu pirmiausia generuojamas trupmeninis Gauso procesas, o jį sumuojant gaunamas trupmeninis Brauno judesys.

Tam, kad būtų išvengta Cholesky dekompozicijos ir daug resursų reiklaujančių veiksmų su matricomis, pasinaudojama trupmeninio Brauno judesio autokovariacijų matricos G:

$$G(i,j) = \gamma(j-i), \qquad i,j = 0,\dots,N-1 \tag{12}$$

savybėmis. Ši matrica priklauso Toeplitzo matricų grupei, todėl $G(i,j) = G(i+1,j+1)$. Tai reiškia, kad iš matricos pirmos eilutės galima rekursiškai išskaičiuoti visą matricą. Levinsono algoritmas skaičiuoja matricą L, kuri tenkina $LL' = G$. Pirmame šio algoritmo žingsnyje pažymėkime:

$$k_1 = -\gamma\left(\frac{1}{N}\right), \qquad \sigma_1 = \sqrt{\gamma(0)}, \tag{13}$$

$$L^1 = \left(1, \gamma\left(\frac{1}{N}\right), ..., \gamma\left(\frac{N-1}{N}\right)\right)^t, \tag{14}$$

$$\hat{L}^1 = \left(0, \gamma\left(\frac{1}{N}\right), ..., \gamma\left(\frac{N-1}{N}\right)\right)^t. \tag{15}$$

Apibrėžiami N ilgio vektoriai:

$$L^j = \sigma_j\left(0, ..., 0, 1, l_2^j, ..., l_{N-j+1}^j\right)^t, \tag{16}$$

$$\hat{L}^j = \sigma_j\left(0, ..., 0, 1, \hat{l}_2^j, ..., \hat{l}_{N-j+1}^j\right)^t. \tag{17}$$

Algoritmo žingsnyje $j+1$:

$$k_{j+1} = \frac{-\hat{l}_1^j}{\sigma_j^2}, \qquad \sigma_{j+1} = \sigma_j(1-k_{j+1}^2), \tag{18}$$

$$\begin{pmatrix} \hat{L}^{j+1} \\ L^{j+1} \end{pmatrix} = \begin{pmatrix} I_N & k_{j+1}Z \\ k_{j+1}I_N & Z \end{pmatrix} \begin{pmatrix} \hat{L}^j \\ L^j \end{pmatrix}, \tag{19}$$

čia

$$Z = \begin{pmatrix} 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}. \tag{20}$$

Baigus vykdyti algoritmą, gaunama matrica $L = (L^1, \cdots, L^N)D^{-1}$, čia

$$D = \begin{pmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_N \end{pmatrix}. \tag{21}$$

L padauginus iš N ilgio nepriklausomų Gauso atsitiktinių dydžių vektoriaus, gaunamas trupmeninis Gauso procesas, kurį sumuojant gaunamas trupmeninis Brauno judesys.

Priede pateikiamas Coeurjolly [6] programinis kodas, perrašytas $\mathbb{R}$ paketo sintakse, kuris generuoja trupmeninį Brauno judesį Levinsono metodu.

4.6. Bangelių metodas

1995m. F. Sellanas ir Y. Meyeris pasiūlė naują algoritmą, kurį po metų įgyvendino ir aprašė P. Abry kartu su pačiu F. Sellanu [1]. Autoriai šį algoritmą vadina bangų sintezę (angl. the wavelet–based synthesis). Pagrindinė idėja yra trupmeninį Brauno judesį išreikšti per sumas:

$$B_t^H = \sum_k \lambda(k)(D^{(-s)}\phi_0)(t-k) + \sum_{j \leqslant 0, k} \gamma_j(k)(D^{(-s)}\psi_{j,k})(t), \tag{22}$$

čia $D^{(s)}$ autoriai žymi s eilės trupmeninį diferencialą. Autoriai pasiūlė kitokią išraišką

$$B_t^H - b_0 = \sum_k b_h(k)\phi_{0,k}^{(s)}(t) + \sum_{j \leqslant 0, k} \gamma_j(k) 4^{-s} 2^{-js} \psi_{j,k}^{(s)}(t), \tag{23}$$

čia $s = H + \frac{1}{2}$, b_0 konstanta, γ_j nepriklausomi vienodai pasiskirstę Gauso atsitiktiniai dydžiai, $b_h(k)$ yra trupmeninis $ARIMA(0, s, 0)$ procesas, $\phi^{(s)}$ tai mastelio (angl. scale) funkcija, o $\psi^{(s)}$ bangos (angl. wavelet). Daugiau informacijos apie šį metodą galima rasti [1] ir [6] šaltiniuose. Naudojamas $\mathbb{R}$ kodas yra iš Coeurjolly darbo [6], jis šiek tiek pakeistas pateikiamas priede.

4.7. Wood – Chano algoritmas

Wood – Chano algoritmas dar yra vadinamas ciklinės matricos metodu. Jis laikomas vienu greičiausiu metodu, skirtu generuoti Gauso procesus. Algoritmas susideda iš kelių etapų. Norint išvengti tiesioginio kovariacijų matricos šaknies skaičiavimo, pirmiausia generuojama ciklinė matrica C:

$$C = \begin{pmatrix} c_0 & c_1 & \dots & c_{m-1} \\ c_{m-1} & c_0 & \dots & c_{m-2} \\ \vdots & \vdots & \ddots & \vdots \\ c_1 & c_2 & \dots & c_0 \end{pmatrix}, \text{ čia } c_j = \begin{cases} \gamma(\frac{j}{N}), \text{ kai } 0 \leqslant j < \frac{m}{2} \\ \gamma(\frac{m-j}{N}), \text{ kai } \frac{m}{2} < j < m-1. \end{cases} \tag{24}$$

$m \geqslant 2(N-1)$, $m \in \mathbb{N}$. C yra išreiškiama per $C = Q\Lambda Q^*$, čia matrica Λ yra, sudaryta iš C tikrinių vektorių, o Q yra unitarinė matrica:

$$Q_{j,k} = m^{-1/2} \exp\left(-2i\pi \frac{jk}{m} \right), \qquad j,k = 0,\dots,m-1. \tag{25}$$

Kadangi Q yra unitarinė, tai $Y = Q\Lambda^{1/2}Q^*Z$ su $Z \sim \mathcal{N}(0, I_m)$, tada $Y \sim \mathcal{N}(0, C)$. Toliau išskiriami trys etapai:

- Naudojant greitąją Furje transformaciją skaičiuojamos C tikrinės reikšmės:

$$\lambda_k = \sum_{j=0}^{m-1} c_j \exp\left(-2i\pi \frac{jk}{m} \right) = \sum_{j=0}^{m-1} c_j \exp\left(2i\pi \frac{jk}{m} \right), k = 0,\dots,m-1; \tag{26}$$

- Generuojama Q^*Z:
  - generuojami nepriklausomi standartiniai normalieji atsitiktiniai dydžiai U ir V, $W_0 = U$, o $W_{\frac{m}{2}} = V$
  - kiekvienam j, $1 \leqslant j < \frac{m}{2}$, generuojama U_j, V_j ir skaičiuojama:

$$W_j = \frac{1}{\sqrt{2}}(U_j + iV_j), \qquad W_{m-j} = \frac{1}{\sqrt{2}}(U_j - iV_j). \tag{27}$$

- Naudojant greitąją Furje transformaciją suskaičiuojama trupmeninio Gauso proceso seka X:

$$X\left(\frac{k}{N}\right) = \frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} \sqrt{\lambda_j} W_j \exp\left(-2i\pi \frac{jk}{m} \right), \qquad k = 0,\dots,m-1. \tag{28}$$

Galiausiai sumuojant X gaunamas trupmeninis Brauno judesys B_H. Daugiau apie šį metodą galima paskaityti [6] ir [4]. Priede pateikiamas realizavimo kodas, parašytas remiantis Coeurjolly [6].

5. Hursto parametro vertinimo testai

Norint nustatyti, kuris generavimo metodas yra geriausias, reikia pasirinkti kriterijų sąrašą, kuriais metodus galima lyginti tarpusavyje. Natūralūs reikalavimai metodams yra sparta ir tikslumas. Spartą matuosime stebėdami kaip keičiasi skaičiavimų trukmė, keičiant generuojamo proceso ilgį. Panašių eksperimentų rezultatai rodo, kad lėčiausi metodai yra stochastinio integralo aproksimavimas ir kovariacinės matricos išdėstymas. Tuo tarpu greičiausiais laikomi metodai, pagrįsti cirkuliacinėmis matricomis. Algoritmų tikslas yra generuoti trupmeninį Brauno judesį su iš anksto parinktu Hursto parametru. Todėl algoritmų tikslumą matuosime vertindami Hursto parametrą tam tikrais metodais ir palygindami jį su parinkta reikšme. Tai reikės atlikti keičiant H ir aibės dydį. Toliau apžvelgsime testus, kurie yra skirti įvertinti Hursto parametrą.

5.1. Hursto parametro pasikliautinieji intervalai

Breton ir Coeurjolly darbe [2] aprašyti trupmeninio Brauno judesio Hursto parametro pasikliautinieji intervalai. Šie pasikliovimo intervalai atrodo taip:

$$\tilde{H}_n^{\inf}(\alpha) := \max\left(0, g_n^{-1}\big(\log\big(x^a_{l,n-l}(\alpha/2)\big) - \log\big(S_n^a\big)\big)\right), \tag{29}$$

$$\tilde{H}_n^{\sup}(\alpha) := \min\left(1, g_n^{-1}\big(\log\big(x^a_{r,n-l}(\alpha/2)\big) - \log\big(S_n^a\big)\big)\right), \tag{30}$$

čia

$$x^a_{l,n-l}(\alpha) := 1 - \frac{q^a_{l,n-l}(\alpha)}{\sqrt{n-l}}, \quad x^a_{r,n-l}(\alpha) := 1 + \frac{q^a_{r,n-l}(\alpha)}{\sqrt{n-l}}, \tag{31}$$

$$q^a_{l,n-l}(\alpha) := (\varphi_{l,n})^{-1}(\alpha;\kappa^a), \quad q^a_{r,n-l}(\alpha) := (\varphi_{r,n})^{-1}(\alpha;\kappa^a). \tag{32}$$

Čia funkcijos $\varphi_{l,n}(\alpha;\kappa^a)$ ir $\varphi_{r,n}(\alpha;\kappa^a)$ apibrėžiamos taip:

$$\varphi_{l,n}(\alpha;\kappa^a) := e^{\frac{\alpha\sqrt{n}}{\kappa^a}}\left(1 - \frac{\alpha}{\sqrt{n}}\right)^{\frac{n}{\kappa^a}} \mathbb{I}_{[0,\sqrt{n}]}(\alpha), \tag{33}$$

$$\varphi_{r,n}(\alpha;\kappa^a) := e^{-\frac{\alpha\sqrt{n}}{\kappa^a}}\left(1+\frac{\alpha}{\sqrt{n}}\right)^{\frac{n}{\kappa^a}} \tag{34}$$

Taip pat, kai $n \to +\infty$:

$$q^a_{l,n-l}(\alpha) \sim q^a_{r,n-l}(\alpha) \sim q^a(\alpha) := \sqrt{2\kappa^a \log(1/\alpha)}. \tag{35}$$

κ^a žymi konstantas prie atitinkamų filtrų, jų yra keletas, platesnę informaciją apie konstantas ir filtrus galima rasti Breton – Coeurjolly straipsnyje [2]. Šiame darbe skaičiavimuose naudojami antros eilės filtrai, todėl $\kappa^a = 2.667$. Antros eilės filtras yra $a = \{1, -2, 1\}, l = 2, p = 2$, pagal 8 apibrėžimą.

$$S^a_n := \frac{1}{n-l}\sum_{i=l}^{n-1} B^a_H\left(\frac{i}{n}\right)^2, \tag{36}$$

čia

$$B^a_H\left(\frac{i}{n}\right) := \sum_{q=0}^{l} a_q B_H\left(\frac{i-q}{n}\right). \tag{37}$$

Liko apibrėžti funkciją g_n:

$$g_n(x) = 2x\log(n) - \log(\pi^a_x(0)), \tag{38}$$

čia $\pi^a_x(0) = -\frac{1}{2}\sum_{q,r=0}^{l} a_q a_r |q-r|^{2x}$. Kadangi skaičiuojant naudojamas antros eilės filtras, todėl $\pi^a_x(0) = 4 - 4^x$.

Trupmeninio Brauno judesio H parametro įvertinys apibrėžiamas taip:

$$\hat{H}^{std}_n(a) := g_n^{-1}(-\log(S^a_n)). \tag{39}$$

Priede pateikiame $\mathbb{R}$ paketo kodą, kurį naudojant galima apskaičiuoti šiuos pasikliautinius intervalus ir H įvertį.

5.2. Karhunen – Loeve Hursto parametro įvertinimo metodas

Karhunen — Loeve metodas aprašytas Salomono ir Forto darbe [11]. Anot autorių, šis metodas savo tikslumu beveik prilygsta didžiausio tikėtinumo metodui, tačiau yra lengviau apskaičiuojamas. Algoritmas paremtas proceso K–L dekompozicija. Tegul $X^{\theta_0}(t)$ yra generuotas trupmeninis Brauno judesys, čia $t = (t_1, t_2, ..., t_n)$, o θ_0 yra Hursto parametras. θ_0 įvertinimo algoritmas atrodo taip:

- Kiekvienam $\theta \in \{0.05, 0.06, ..., 0.95\}$ yra suskaičiuojamas $Y^{\theta,\theta_0}(t) = (\Sigma^\theta)^{-1/2} X^{\theta_0}(t)$;

- Kiekvienu atveju suskaičiuojama $h_n(\theta;\theta_0) = \frac{\mathbb{E}\|Y^{\theta,\theta_0}(t)\|^2}{n}$;
- Skaičiuojamas Hursto parametro įvertis $\hat{\theta}_0 = \arg\min_\theta |h_n(\theta,\theta_0) - 1|$.

Čia Σ^θ žymi trupmeninio Brauno judesio su parametru θ kovariacijų matricą, kuri suskaičiuojama naudojant Γ (iš 10 formulės) tikrines reikšmes λ_k ir tikrinius vektorius φ_k:

$$\Sigma^\theta = \sum_{k\geqslant 1} \lambda_k \varphi_k(t_i)\varphi_k(t_j), \quad 1 \leqslant i,j \leqslant n. \tag{40}$$

Nors ir teigiama, kad šis algoritmas tikslus, prilygstantis didžiausio tikėtinumo metodui, visgi jis yra pernelyg lėtas. Įverčiui suskaičiuoti reikia 94 kartus generuoti Γ matricą, skaičiuoti jos tikrines reikšmes ir tikrinius vektorius, tiek pat kartų skaičiuoti matricą Σ^θ, kurią dar reikia pakelti $-1/2$ laipsniu. Visi šie veiksmai atima daug skaičiavimo laiko, algoritmas veikia lėtai vos padidinus proceso realizacijos ilgį. Šio algoritmo realizavimas $\mathbb{R}$ paketu pateikiamas priede.

5.3. Logaritminės periodogramos metodas

Tegul $1 \leqslant m_1 \leqslant m_2 \leqslant N^* = [N - 1/2]$. Logaritminių periodogramų (toliau log – periodogramų) metodu gaunamas H įvertis yra lygus:

$$\hat{H}_N(m_1,m_2) = \frac{1}{2}(1 - \hat{\alpha}_N), \tag{41}$$

čia $\hat{\alpha}_N$ yra regresijos tarp $\big\{\log(I_N(\lambda_k))\big\}_{m_1\leqslant k\leqslant m_2}$ ir $\big\{\log(\lambda_k)\big\}_{m_1\leqslant k\leqslant m_2}$ įvertis.

$$I_N(\lambda) = \frac{1}{2\pi N}\left|\sum_{t=0}^{N-1} X(t)e^{-it\lambda}\right|^2, \quad \lambda = \lambda_{k,N} = \frac{2\pi k}{N}. \tag{42}$$

Regresija įvertinama Mažiausių kvadratų metodu, o naudojant gautą įvertį $\hat{\alpha}_N$, suskaičuojamas H įvertis. Daugiau apie šį metodą galima rasti [6], priede pateikiamas Coeurjolly paremtas $\mathbb{R}$ paketo programos kodas, kuris skaičiuoja anksčiau aprašytą įvertį.

5.4. Lobato ir Robinsono metodas

Tegul $F(\lambda) = \int_0^\lambda f(\theta)d\theta$, tada nulio aplinkoje egzistuoja log – tiesinis ryšys tarp dviejų $F(\lambda)$ reikšmių [6]. Tegul $q \in (0;1)$, tada $\frac{F(q\lambda)}{F(\lambda)} \to q^{2-2H}$, $|\lambda| \to 0$. H įvertis randamas taip:

$$\hat{H}_N(q,m_1,m_2) = 1 - \frac{1}{2\log(q)}\log\left(\frac{\hat{F}(q\lambda_{m_2})}{\hat{F}(\lambda_{m_2})}\right), \tag{43}$$

čia

$$\hat{F}(\lambda_k) = \frac{2\pi}{N} \sum_{j=1}^{[N\lambda_{k,N}/2\pi]} I(\lambda_j), \quad k = m_1, ...m_2. \tag{44}$$

Šis ir log – periodogramų metodas yra priskiriami prie spektrinių vertinimo algoritmų. Kaip parodys tolimesni skaičiavimai, šiais metodais gauti rezultatai labai panašūs. Priede pateikiamas $\mathbb{R}$ paketo programos kodas.

5.5. Diskrečios variacijos metodas

Hursto parametro įvertinimui pasirinkome pirmos ir antros eilės diskrečios variacijos metodus. Šie algortimai nagrinėti D. Melichovo darbe [10], laikomi gana tiksliais ir greitais H parametro vertinimo algoritmais. Tegul $X = \{X_t; t \in [0;1]\}$. Pirmu atveju įvertis apskaičiuojamas taip:

$$\hat{H}^n_{dv1} = \frac{1}{2} - \frac{1}{2\ln 2} \ln \frac{V^{(1)}_{2n}(X,2)}{V^{(1)}_n(X,2)}. \tag{45}$$

Antru atveju:

$$\hat{H}^n_{dv2} = \frac{1}{2} - \frac{1}{2\ln 2} \ln \frac{V^{(2)}_{2n}(X,2)}{V^{(2)}_n(X,2)}. \tag{46}$$

$V^{(1)}_n(X,2)$ žymi pirmos eilės kvadratines variacijas, o $V^{(2)}_n(X,2)$ — antros eilės kvadratines variacijas. Jos apibrėžiamos taip:

$$V^{(1)}_n(X,2) = \sum_{k=1}^{n} (\Delta^{(1)}_k X)^2, \quad V^{(2)}_n(X,2) = \sum_{k=1}^{n-1} (\Delta^{(2)}_k X)^2. \tag{47}$$

Čia

$$\Delta^{(1)}_k X = X(t^n_k) - X(t^n_{k-1}), \quad \Delta^{(2)}_k X = X(t^n_{k+1}) - 2X(t^n_k) + X(t^n_{k-1}). \tag{48}$$

$V^{(\cdot)}_{2n}(X,2)$ žymi kvadratinę variaciją ne pilnai sekai, o poaibiui $\{X_k : k = 2j, 0 \leqslant j \leqslant [n/2]\}$. Šių algoritmų realizavimas $\mathbb{R}$ paketu yra pagal D. Melichovą [10], kodas pateikiamas priede.

5.6. Diskrečios variacijos metodas (MKM)

Kaip ir anksčiau aprašyti metodai tai yra diskrečios variacijos metodas, kuris veikia panaudojant mažiausių kvadratų metodą (MKM). Šio algoritmo realizavimo kodas parašytas pagal Coeurjolly [6]. Kadangi šis algoritmas

laikomas vienu geriausiu [6], jį vėliau palyginsime su kitais metodais. Apibrėžiama filtrų seka a^m, $1 \leqslant m \leqslant M$:

$$a_i^m = \begin{cases} a_j, \text{ kai } i = jm \\ 0, \text{ kitu atveju} \end{cases}, \qquad i = 0, \ldots, ml + 1. \tag{49}$$

Tada $\mathbb{E}(S_N(k, a^m)) = m^{Hk}\mathbb{E}(S_N(k, a))$. Čia:

$$S_N(k, a) = \frac{1}{N - l} \sum_{i=l}^{N-1} \left| V^a\left(\frac{i}{N}\right) \right|^k, \qquad k > 0, \tag{50}$$

$$V^a\left(\frac{i}{N}\right) = \sum_{q=0}^{l} a_q B_{H,C}\left(\frac{i - q}{N}\right), \qquad \forall i = \{l, \ldots, N - 1\}. \tag{51}$$

Toliau galima pažymėti, kad $S_N(k, a^m)$ yra $\mathbb{E}(S_N(k, a^m))$ įvertinys, o tokiu atveju H galima įvertinti naudojant regresiją tarp $\log S_N(k, a^m)$ ir $k \log(m)$, $1 \leqslant m \leqslant M$. Regresijos įvertinimas leidžia gauti $\hat{H}$ įvertį. Šiuo atveju algoritmas realizuotas naudojant mažiausių kvadratų metodą.

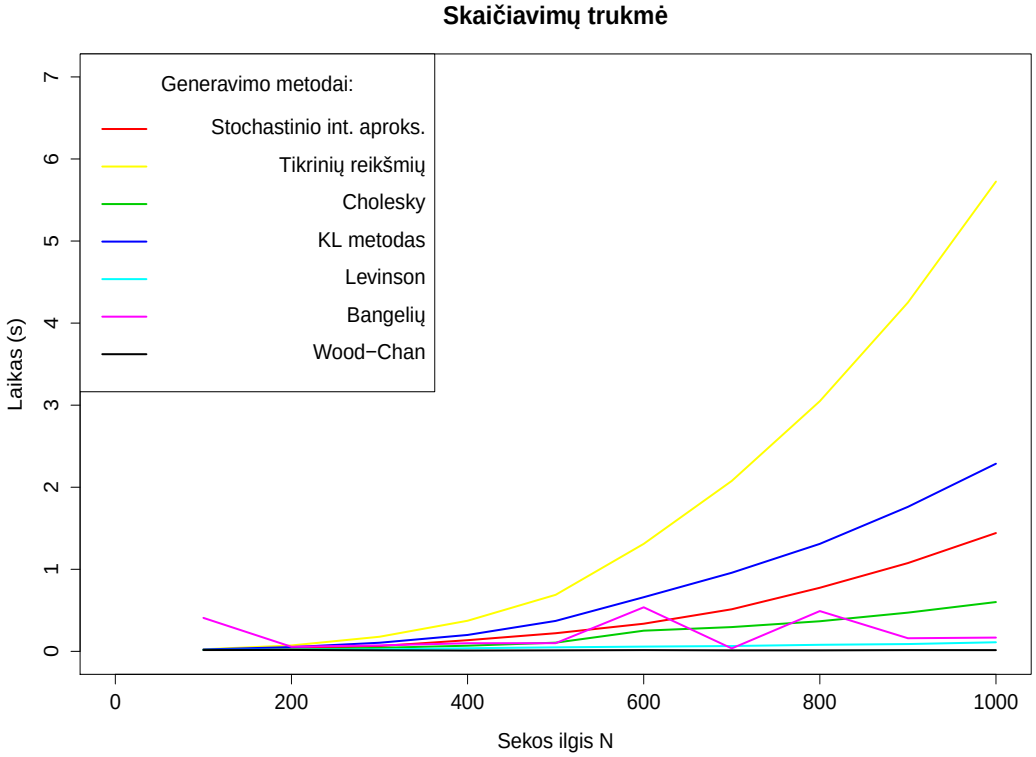
6. Modeliavimas ir testavimas

Generuojant procesą labai svarbu gauti norimų parametrų realizacijas. Be to norima tai alikti greitai ir kokybiškai. Šioje dalyje lyginsime algoritmus tarpusavyje ir pamėginsime išrinkti geriausią. Užduočiai atlikti reikės palyginti ir Hursto parametro vertinimo metodus, geriausias iš jų bus naudojamas generavimo algoritmų tikslumui nustatyti.

6.1. Generavimo algoritmų sparta

Greičio palyginimą atliekame keliais etapais. Kadangi atlikdami eksperimentus pastebime, kad kai kurie algoritmai skaičiuoja labai greitai, o kiti yra žymiai lėtesni, pirmame etape lyginsime generavimo greitį, kai elementų kiekis N auga nuo 100 iki 1000. Generavimo trukmė neturėtų priklausyti nuo H parametro, todėl lygindami algoritmų spartą visais atvejais fiksuojame $H = 0.5$. Kadangi daug laiko atima matricų skaičiavimai, įvairūs ciklai, bendra algoritmų realizavimo logika, o nuo visų šių veiksnių labai priklauso skaičiavimų efektyvumas, tai tarsime, kad suprogramuoti algoritmai yra pakankamai optimalūs. Pasinaudojame $\mathbb{R}$ pakete esama funkcija `system.time`, kuri leidžia išmatuoti trukmę, kurią kompiuteris užtrunka atlikdamas skaičiavimus. Kiekvienu atveju skaičiavimai atliekami po 20

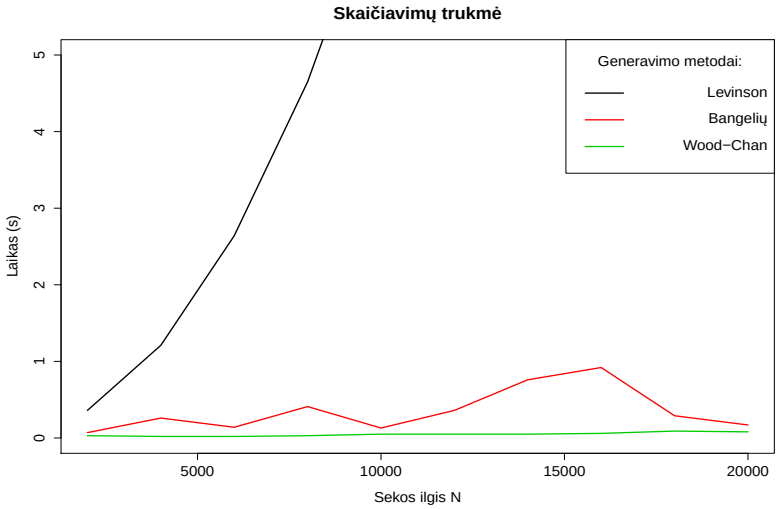
kartų ir išvedamas sugaišto laiko vidurkis kiekvienam N. Remdamiesi šia analize, galime daryti preliminarias išvadas, kad greičiausi algoritmai yra Levinsono, Wood – Chano ir bangelių — visi, kuriems nereikia tiesiogiai generuoti kovariacijų matricų. Iš lėtesnių algoritmų greičiausias šiuo atveju yra Cholesky, jei reikia generuoti 1000 ilgio seką, naudojant šį algoritmą laukti reikės dalį sekundės. Gana įdomūs rezultatai gauti generavimui taikant bangelių metodą. Jei praktiškai visų algoritmų generavimo sparta augo didinant N, tai bangelių metodo atveju skaičiavimų trukmė svyravo. Pirmo etapo rezultatus pateikiame 1 paveiksle.



1 pav. Generavimo trukmė sekundėmis, didėjant N

Antrame etape palyginsime tris greičiausius algoritmus: Wood – Chano, bangelių ir Levinsono. Kaip ir pirmame etape fiksuojame $H = 0.5$, viską kartojame po 20 kartų ir vedame laiko vidurkius, tačiau dabar N keičiame nuo 2000 iki 20000. Iš gautų rezultatų matyti, kad greičiausias generavimo algoritmas yra Wood – Chano. Tuo tarpu bangelių metodas nėra stabilus laiko prasme — didinant N, skaičiavimų trukmė kinta įvairiai. Levinsono algoritmas, esant dideliems N, skaičiuoja ilgiausiai iš šių trijų metodų. Gautus eksperimento rezultatus pateikiame 2 paveiksle.

Tam, kad galutinai įsitikintume, kuris algoritmas yra greičiausias, kartojame eksperimentą su bangelių ir Wood – Chano algoritmais. Fiksuojame $H = 0.5$, darome analogišką procedūrą, tik N didiname nuo 3000 iki 30000. Po šio eksperimento tapo akivaizdu, kad Wood – Chano yra geriausias algoritmas skaičiavimo laiko prasme. Visais atvejais, naudojant Wood – Chano metodą, skaičiavimams pakako mažiau nei 1 sekundės. Bangelių algoritmo skaičiavimų trukmė svyruoja, kartais viršija ir 6 sekundes. Rezultatus pateikiame 3 paveiksle.



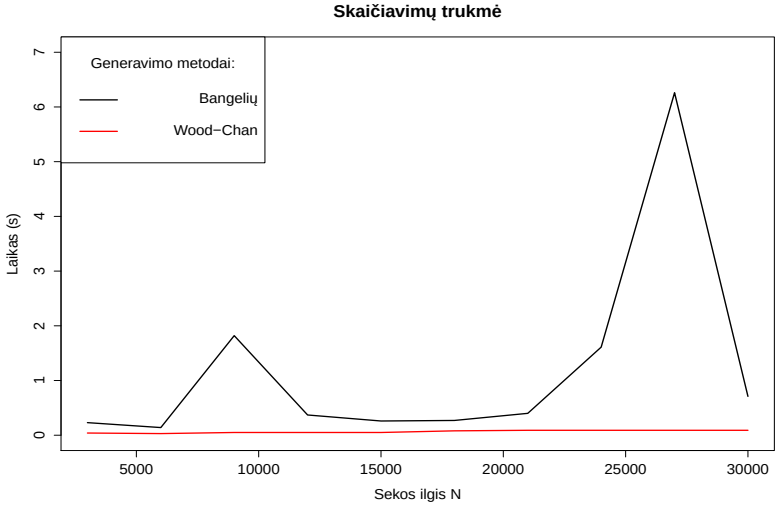
2 pav. Generavimo trukmė sekundėmis, didėjant N nuo 2000

6.2. Gausiškumo savybė

Trupmeninis Brauno judesys priklauso Gauso procesų šeimai. Dėl šios priežasties procesas kiekvienu laiko momentu turi turėti Gauso skirstinį. Gausiškumą patikrinsime naudodami Kolmogorovo – Smirnovo testą (toliau — K–S testas). Keisdami sekų ilgį N (30, 100 ir 500) atliksime tokį eksperimentą:

- 100 kartų generuojame N ilgio trupmeninį Brauno judesį (B_t);
- Kiekvienam $t \in [1 : N]$ taikome K–S testą, norėdami patikrinti, ar tai yra normalusis skirstinys. Tikrinama pirma hipotezė:

$$\begin{cases} H_0 : B_t \sim \mathcal{N}(\mu, \sigma^2), \\ H_1 : B_t \not\sim \mathcal{N}(\mu, \sigma^2). \end{cases} \tag{52}$$



3 pav. Generavimo trukmė sekundėmis, didėjant N nuo 3000

Išsaugome testo p–reikšmes p_t, čia $t \in [1 : N]$.

- Naudodami K–S testą patikriname, ar p–reikšmės p_t turi tolygųjį skirstinį. Tikrinama hipotezė:

$$\begin{cases} H_0 : p_t \sim \mathcal{U}(0;1), \\ H_1 : p_t \not\sim \mathcal{U}(0;1). \end{cases} \tag{53}$$

Išsaugome testo p–reikšmes.

Šią procedūrą kartojame visiems generavimo algoritmams, keisdami H parametro reikšmes $(0.1, 0.2, ..., 0.9)$. Remiamės faktu, kad nulinė hipotezė yra teisinga tada ir tik tada, kai p–reikšmės p_t turi tolygų skirstinį. Vadinasi, jei K–S testo p–reikšmės (53) yra didesnės už 0.05, tai su reikšmingumo lygmeniu $\alpha = 0.05$ priimame H_0 (52), o iš to išplaukia, kad generuotos sekos turi Gauso skirstinį. Atlikę aprašytą procedūrą, gavome, kad gausiškumo savybę geriausiai tenkina sekos, kurios buvo generuotos naudojant tikrinių reikšmių, Cholesky, Karhunen – Loeve ir Wood – Chano algoritmus. Tiesa, kai $N = 500$ gauti rezultatai buvo prastesni, šiuo atveju beveik visos p–reikšmės buvo mažesnės už reikšmingumo lygmenį $\alpha = 0.05$. Rezultatai pateikiami lentelėse: 1, 2, 3.

1 lentelė. Gausiškumo tenkinimas ($N = 30$). Hipotezės (53) p–reikšmės.

H	St.int.	Tikr.r.	Chol	KL	Levin.	Bangel.	Wood-Chan
0.1	0.00	0.44	0.20	0.45	0.08	0.00	0.07
0.2	0.00	0.01	0.19	0.59	0.00	0.00	0.12
0.3	0.00	0.97	0.00	0.00	0.00	0.00	0.00
0.4	0.00	0.05	0.06	0.41	0.00	0.00	0.13
0.5	0.00	0.10	0.14	0.03	0.00	0.00	0.05
0.6	0.00	0.09	0.69	0.59	0.00	0.00	0.62
0.7	0.00	0.05	0.86	0.72	0.00	0.00	0.01
0.8	0.00	0.01	0.53	0.00	0.00	0.00	0.07
0.9	0.00	0.01	0.85	0.14	0.00	0.00	0.30
Viso	0.00	0.19	0.39	0.33	0.01	0.00	0.15

2 lentelė. Gausiškumo tenkinimas ($N = 100$). Hipotezės (53) p-reikšmės.

H	St.int.	Tikr.r.	Chol	KL	Levin.	Bangel.	Wood-Chan
0.1	0.00	0.01	0.38	0.24	0.00	0.00	0.00
0.2	0.00	0.88	0.00	0.00	0.00	0.00	0.01
0.3	0.00	0.63	0.11	0.00	0.00	0.00	0.18
0.4	0.00	0.00	0.29	0.00	0.00	0.00	0.00
0.5	0.00	0.08	0.13	0.00	0.00	0.00	0.84
0.6	0.00	0.76	0.65	0.00	0.00	0.00	0.00
0.7	0.00	0.55	0.09	0.26	0.00	0.00	0.39
0.8	0.00	0.00	0.92	0.00	0.00	0.00	0.00
0.9	0.00	0.00	0.79	0.00	0.00	0.00	0.00
Viso	0.00	0.32	0.37	0.06	0.00	0.00	0.16

3 lentelė. Gausiškumo tenkinimas ($N = 500$). Hipotezės (53) p-reikšmės.

H	St.int.	Tikr.r.	Chol	KL	Levin.	Bangel.	Wood-Chan
0.1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.2	0.00	0.06	0.64	0.00	0.00	0.00	0.04
0.3	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.4	0.00	0.00	0.00	0.00	0.00	0.00	0.05
0.5	0.00	0.00	0.00	0.01	0.00	0.00	0.06
0.6	0.00	0.00	0.11	0.00	0.00	0.00	0.00
0.7	0.00	0.01	0.27	0.00	0.00	0.00	0.00
0.8	0.00	0.00	0.01	0.00	0.00	0.00	0.79
0.9	0.00	0.10	0.00	0.00	0.00	0.00	0.00
Viso	0.00	0.02	0.11	0.00	0.00	0.00	0.11

6.3. H parametro vertinimo testų sparta

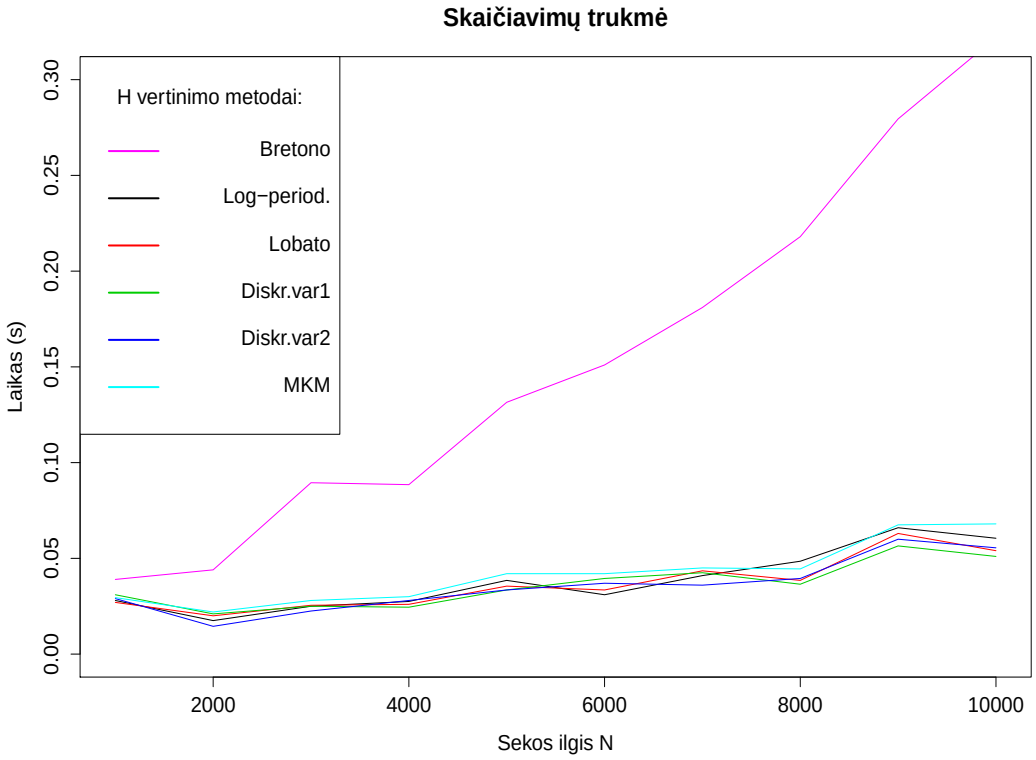
Yra sukurta nemažai algoritmų, procesų generavimui, tačiau ne ką mažiau yra ir testų, skirtų Hursto parametrui H įvertinti. Empiriškai panagrinėsime, ar testai pakankamai tiksliai vertina H parametrą, išrinksime, kuris testas yra geriausias, ir jį vėliau pritaikysime generavimo algoritmų palyginime. Visgi nepaisant tikslumo, labai svarbu ir skaičiavimų atlikimo greitis, todėl pirmiausia palyginsime šių vertinimo algoritmų spartą.

Testavimui naudojame Wood – Chano generavimo algoritmą, kadangi anksčiau gavome, kad jis yra greičiausias. Po 20 kartų generuojame 30, 20,..., 100 ilgio trajektorijas ir vertiname H parametrą. Rezultatus pateikiame 4 lentelėje. Matome, kad vertinant Karhunen – Loeve (K–L) algoritmu skaičiavimai užtrunka ilgiausiai, netgi santykinai nedidelei $N = 100$ skaičiavimai užtruko vidutiniškai 14 sekundžių. Tuo tarpu kiti algoritmai veikė greičiau — laukti nereikėjo nė sekundės.

Pamėginkime palyginti alogritmus dar kartą, kai N kinta nuo 1000 iki 10000. Šį kartą K–L metodo neįtraukiame, nes kaip matėme jis yra itin lėtas. Gavome, kad Bretono testu skaičiavimai užtruko ilgiausiai lyginant su kitais testais (4 grafikas). Kai $N = 10000$, skaičiavimai šiuo testu užtruko vidutiniškai 0.32(s), tuo tarpu kitais maždaug 0.05(s). Vis dėlto tai nėra ilgas laiko tarpas, todėl laiko trukmė nebus pagrindinis gerumo kriterijus. Toliau tikrinsime, kuris metodas tiksliau vertina H parametrą.

4 lentelė. H **vertinimo sparta, keičiant** N**. Wood – Chano generavimo metodas** $H = 0.5$**,** $k = 20$

N	Breton	K-L	Log-period.	L-R	MKM	Diskr.var.1	Diskr.var.2
30	0.016	1.092	0.010	0.009	0.014	0.009	0.010
40	0.006	1.954	0.009	0.009	0.012	0.010	0.013
50	0.007	3.078	0.008	0.011	0.010	0.007	0.008
60	0.012	4.523	0.011	0.011	0.013	0.012	0.008
70	0.006	6.260	0.008	0.008	0.011	0.011	0.010
80	0.009	8.431	0.009	0.011	0.010	0.011	0.012
90	0.016	11.012	0.009	0.011	0.010	0.009	0.011
100	0.008	14.024	0.012	0.012	0.010	0.005	0.011



4 pav. H vertinimo trukmė sekundėmis, didėjant N

6.4. H parametro vertinimo testų tikslumas

Skaičiavimus atliksime keliais būdais, keisdami H ir keisdami sekų ilgį N. Naudosime du generavimo algoritmus, kurie buvo geriausi gausiškumo testavime: Cholesky ir Wood – Chano. Pirmiausia fiksuojame $H = 0.5$ ir atliekame tokią procedūrą:

- Generuojame 500 ilgio seką;
- Traukiame tolygiai pasiskirsčiusias 250, 166, 125, 100 ir 83 ilgio imtis;
- Kiekvienai sekai visais testais vertiname H parametrą.

Procedūrą kartojame 50 kartų naudodami Cholseky ir Wood – Chano algoritmus. Gautiems H įverčiams suskaičiuojame vidurkius $\hat{H}$ ir standartinius nuokrypius $\hat{\sigma}$ (5 ir 8 lentelės). Visais atvejais gavome nepaslinktus įverčius lygius $\hat{H} = 0.5$, tačiau mažiausi standartiniai nuokrypiai gauti vertinant Bretono testu. Procedūrą kartojame, šį kartą fiksavę $H = 0.1$. Dabar jau log–periodogramų ir Lobato – Robinsono testais gauti įverčiai buvo paslinkti, neigiami (6 ir 9 lentelės). Mažiausi standartiniai nuokrypiai vėl gauti vertinant Bretono metodu.

Dabar fiksuojame sekų ilgį $N = 500$ ir keičiame $H = 0.1, 0.2, ..., 0.9$. Atliekame procedūrą:

- Kiekvienam H generuojame 500 ilgio seką;
- Kiekvienai sekai visais testais vertiname H parametrą.

Skaičiavimus atliekame Cholesky ir Wood – Chano generavimo metodais, kartojame 50 kartų, skaičiuojame vidurkius $\hat{H}$ ir standartinius nuokrypius $\hat{\sigma}$. Prasčiausi rezultatai (7 ir 10 lentelės) gauti logaritminių periodogramų ir Lobato – Robinsono testais. Pakankamai geri rezultatai gauti tiek mažiausių kvadratų, tiek diskrečių variacijų metodais. Visgi mažiausi nuokrypiai ir vėl gauti vertinant Breton testu. Šį testą laikysime geriausiu iš pasirinktųjų ir naudosime tolimesnėje analizėje vertindami generavimo algoritmus.

5 lentelė. H **vertinimas, keičiant** N**. Wood – Chano metodas** $H = 0.5$**,** $k = 50$

	Breton		Log-period.		L-R		MKM		Diskr.var.1		Diskr.var.2	
N	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$
83	0.5	0.02	0.5	0.16	0.5	0.10	0.5	0.12	0.5	0.11	0.5	0.19
100	0.5	0.02	0.5	0.12	0.5	0.09	0.5	0.10	0.5	0.10	0.5	0.17
125	0.5	0.01	0.5	0.12	0.5	0.09	0.5	0.09	0.5	0.11	0.4	0.18
166	0.5	0.01	0.5	0.10	0.5	0.07	0.5	0.06	0.5	0.08	0.5	0.12
250	0.5	0.01	0.5	0.07	0.5	0.06	0.5	0.05	0.5	0.07	0.5	0.10
500	0.5	0.01	0.5	0.05	0.5	0.04	0.5	0.04	0.5	0.05	0.5	0.08

6 lentelė. H **vertinimas, keičiant** N**. Wood – Chano metodas** $H = 0.1$**,** $k = 50$

	Breton		Log-period.		L-R		MKM		Diskr.var.1		Diskr.var.2	
N	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$
83	0.1	0.02	-0.1	0.14	-0.1	0.16	0.1	0.08	0.1	0.17	0.1	0.26
100	0.1	0.02	-0.1	0.12	-0.2	0.15	0.1	0.06	0.1	0.15	0.1	0.19
125	0.1	0.02	-0.1	0.13	-0.1	0.12	0.1	0.06	0.1	0.15	0.1	0.22
166	0.1	0.02	-0.1	0.10	-0.1	0.12	0.1	0.06	0.1	0.10	0.1	0.15
250	0.1	0.01	-0.1	0.08	-0.1	0.10	0.1	0.04	0.1	0.10	0.1	0.14
500	0.1	0.01	-0.1	0.05	-0.1	0.06	0.1	0.03	0.1	0.08	0.1	0.11

7 lentelė. H **vertinimas. Wood – Chano metodas** $N = 500$**,** $k = 50$

	Breton		Log-period.		L-R		MKM		Diskr.var.1		Diskr.var.2	
H	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$
0.1	0.10	0.01	-0.06	0.05	-0.08	0.05	0.10	0.03	0.10	0.07	0.10	0.10
0.2	0.20	0.01	0.11	0.05	0.09	0.05	0.20	0.03	0.19	0.07	0.19	0.09
0.3	0.30	0.01	0.24	0.05	0.22	0.04	0.29	0.04	0.28	0.06	0.27	0.10
0.4	0.40	0.01	0.38	0.04	0.38	0.05	0.41	0.05	0.41	0.05	0.41	0.08
0.5	0.50	0.01	0.49	0.04	0.50	0.04	0.50	0.04	0.51	0.04	0.51	0.08
0.6	0.60	0.00	0.60	0.04	0.60	0.05	0.59	0.05	0.59	0.04	0.58	0.07
0.7	0.70	0.00	0.72	0.04	0.72	0.03	0.70	0.04	0.69	0.03	0.70	0.06
0.8	0.80	0.01	0.84	0.05	0.81	0.03	0.79	0.04	0.78	0.03	0.79	0.06
0.9	0.90	0.00	0.96	0.05	0.90	0.04	0.90	0.05	0.87	0.03	0.90	0.07

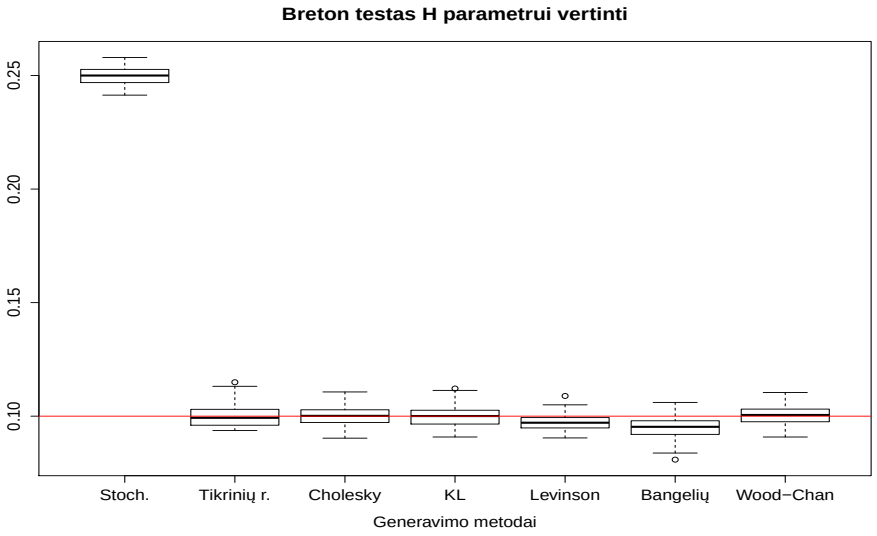
6.5. H parametro vertinimas

Šioje dalyje panagrinėsime, kurie algoritmai tiksliai generuoja trupmeninį Brauno judesį priklausomai nuo parinkto Hursto parametro H. Parinksime skirtingus H ir sugeneravę tBj skirtingais algoritmais Bretono testu bandysime gauti H įverčius. Kadangi anksčiau nustatėme, jog Bretono testas yra labai tikslus, todėl gavę nuokrypius, laikysime, kad prastas generavimo algoritmas. Fiksuojame H ir 100 kartų atliekame tokią procedūrą:

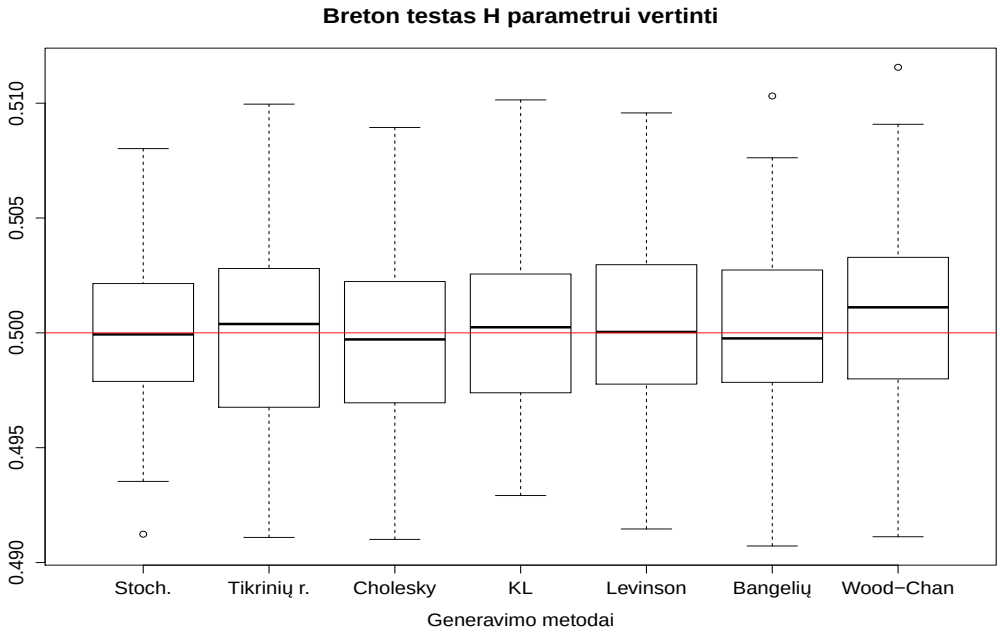
- Kiekvienu algoritmu generuojame 1000 ilgio seką;
- Bretono testu vertiname H įvertį.

Prie $H = 0.1$ prasčiausias buvo stochastinio integralo aproksimavimo algoritmas (5 paveikslas). Geriausiais laikytume Cholesky, Wood – Chano, Karhunen – Loeve ir tikrinių reikšmių algoritmus. Levinson ir bangelių metodais brėžtų sekų H įverčiai buvo šiek tiek paslinkti.

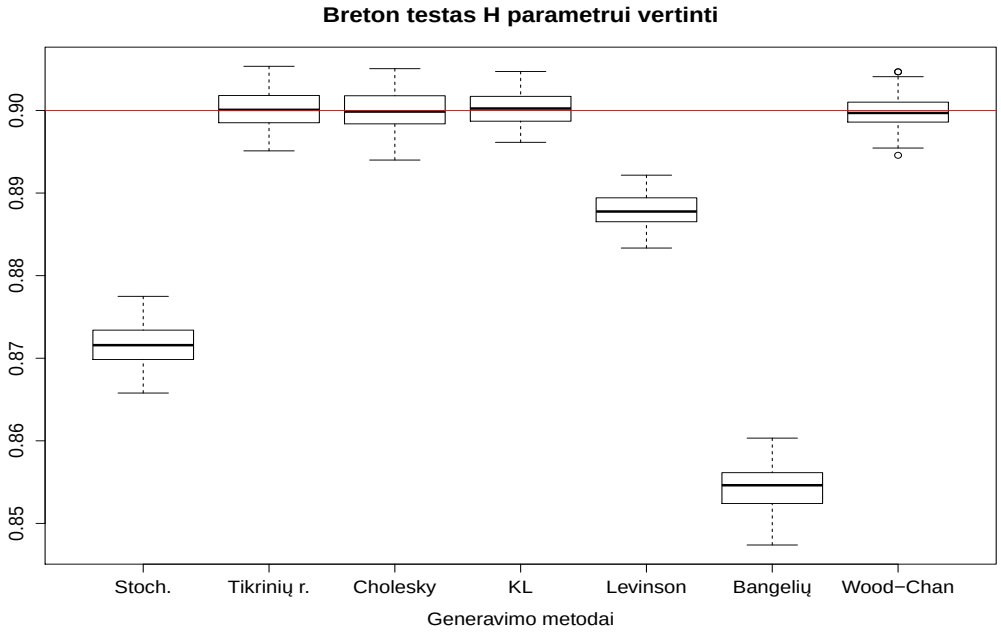
Tą pačią procedūrą kartojame kai $H = 0.5$, gauname, kad visi generavimo algoritmai šiuo atveju veikia pakankamai gerai (6 paveikslas). Dar kartą atlikę procedūrą su $H = 0.9$, matome (7 paveikslas), kad nepaslinktus įverčius gavome tik tikrinių reikšmių, Cholesky, Karhunen – Loeve ir Wood – Chano algoritmais generuotoms sekoms. Bangelių, Levinsono ir stochastinio integralo algoritmai generavo ne tai ko reikėjo, jų H parametrų įverčiai buvo paslinkti.



5 pav. H **vertinimas Breton testu.** $H = 0.1$**,** $N = 1000$**,** $k = 100$



6 pav. H **vertinimas Breton testu.** $H = 0.5$**,** $N = 1000$**,** $k = 100$



7 pav. H **vertinimas Breton testu.** $H = 0.9$**,** $N = 1000$**,** $k = 100$

7. Išvados

Kai $N \to \infty$, greičiausias generavimo metodas yra Wood – Chano algoritmas. Nuo jo nedaug, bet atsilieka bangelių algoritmas. Lėčiausiai veikia algoritmai, kurie grindžiami sudėtingais matricų skaičiavimais, veiksmais su jomis. Vis dėlto rengiant šį darbą pavyko patobulinti žinomą Cholesky algoritmą pašalinus vieną ciklą, taip pagreitinant jo veikimą apie 10 kartų. Taip pat reikėtų pažymėti, kad dirbant su sekomis iki 500 ilgio ($N = 500$), visi metodai užtrunka panašiai - iki 1(s).

Gausiškumo savybę labiausiai tenkina Cholesky, Wood – Chano metodais generuojamos proceso realizacijos. Keičiant tiek H parametrą, tiek sekos ilgį N ir generuojant sekas šiais metodais: stochastinio integralo aproksimavimo, Levinsono ir bangelių, nė karto nepriimta hipotezė H_0, kad generuota seka yra Gauso. Naudotas reikšimingumo lygmuo $\alpha = 0.05$.

Lyginant H vertinimo testus nustatyta, kad visais metodais skaičiavimai užtrunka panašiai, išskyrus Karhunen – Loeve, kuris yra labai lėtas. Kiti metodai net ir esant pakankamai dideliam $N = 10000$ užtrunka iki 1 (s). Pakankamai greitas ir labai tikslus metodas yra Breton – Coeurjolly. Kiti neblogi metodai tikslumo prasme yra diskrečių variacijų metodai. Pastebėta, kad daugelis vertinimo ir generavimo metodų geriau veikia, kai H arti 0.05, o rezultatai suprastėja, kai H artėja prie 0 ar 1.

Generuojant ir vertinant H parametro reikšmę, mažiausi nuokrypiai gauti naudojant tikrinių reikšmių, Cholesky, Karhunen – Loeve ir Wood – Chano algoritmus. Didžiausi nuokrypiai buvo generuojant stochastinio integralo aproksimavimo, Levinsono ir bangelių metodais. H vertinimas buvo vykdomas naudojant Breton – Coeurjolly metodą.

Apibendrinus rezultatus daroma išvada, kad visgi geriausias generavimo metodas yra Wood – Chano. Dirbant su neilgomis sekomis galima naudoti ir Cholesky metodą, kuris yra pakankamai tikslus, tik lėtesnis.

Priedai

Lentelės

8 lentelė. H **vertinimas, keičiant** N**. Cholesky metodas** $H = 0.5$**,** $k = 50$

	Breton		Log-period.		L-R		MKM		Diskr.var.1		Diskr.var.2	
N	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$
83	0.5	0.02	0.5	0.16	0.5	0.10	0.5	0.10	0.5	0.09	0.5	0.14
100	0.5	0.02	0.5	0.15	0.5	0.10	0.5	0.09	0.5	0.10	0.5	0.15
125	0.5	0.01	0.5	0.13	0.5	0.09	0.5	0.07	0.5	0.09	0.5	0.15
166	0.5	0.01	0.5	0.11	0.5	0.07	0.5	0.07	0.5	0.09	0.5	0.15
250	0.5	0.01	0.5	0.08	0.5	0.05	0.5	0.06	0.5	0.07	0.5	0.11
500	0.5	0.00	0.5	0.05	0.5	0.04	0.5	0.04	0.5	0.04	0.5	0.08

9 lentelė. H **vertinimas, keičiant** N**. Cholesky metodas** $H = 0.1$**,** $k = 50$

	Breton		Log-period.		L-R		MKM		Diskr.var.1		Diskr.var.2	
N	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$
83	0.1	0.02	-0.1	0.17	-0.1	0.17	0.1	0.08	0.1	0.17	0.1	0.22
100	0.1	0.02	-0.2	0.15	-0.2	0.13	0.1	0.05	0.1	0.13	0.1	0.18
125	0.1	0.02	-0.1	0.11	-0.1	0.12	0.1	0.06	0.1	0.14	0.1	0.18
166	0.1	0.01	-0.1	0.11	-0.1	0.10	0.1	0.04	0.1	0.11	0.1	0.14
250	0.1	0.01	-0.1	0.07	-0.1	0.08	0.1	0.04	0.1	0.09	0.1	0.12
500	0.1	0.01	-0.1	0.05	-0.1	0.06	0.1	0.03	0.1	0.08	0.1	0.10

10 lentelė. H **vertinimas. Cholesky generavimo metodas** $N = 500$**,** $k = 50$

	Breton		Log-period.		L-R		MKM		Diskr.var.1		Diskr.var.2	
H	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$	$\hat{H}$	$\hat{\sigma}$
0.1	0.10	0.01	-0.07	0.03	-0.09	0.06	0.10	0.03	0.11	0.08	0.11	0.10
0.2	0.20	0.01	0.09	0.05	0.08	0.06	0.20	0.04	0.20	0.06	0.20	0.08
0.3	0.30	0.01	0.25	0.05	0.23	0.05	0.30	0.03	0.30	0.06	0.30	0.09
0.4	0.40	0.01	0.37	0.04	0.37	0.04	0.40	0.04	0.39	0.05	0.39	0.07
0.5	0.50	0.01	0.50	0.05	0.50	0.04	0.50	0.04	0.51	0.05	0.52	0.08
0.6	0.60	0.01	0.61	0.05	0.61	0.04	0.60	0.04	0.60	0.04	0.62	0.08
0.7	0.70	0.00	0.73	0.04	0.72	0.03	0.70	0.04	0.69	0.04	0.68	0.08
0.8	0.80	0.00	0.84	0.04	0.82	0.04	0.79	0.04	0.79	0.04	0.80	0.06
0.9	0.90	0.00	0.97	0.05	0.91	0.04	0.89	0.05	0.88	0.03	0.89	0.06

R programinis kodas

Generavimo algoritmai.

- Stochastinio integralo aproksimavimo metodas. Remiantis Coeurjolly [6].

```
fBmMvn=function(H,n){
  if(missing(n)) n=500
  if(missing(H)) H=0.6

  db1=rnorm(n)
  borne=trunc(n^1.5)
  db2=rnorm(borne)
  fBm=rep(0,n)
  CH=sqrt(gamma(2*H+1)*sin(pi*H))/gamma(H+1/2)

  ind1=(1:n)^(H-1/2)
  ind2=(1:(borne+n))^(H-1/2)
  for(i in (2:n)){
    I1=db1[(i-1):1]*ind1[1:(i-1)]
    I2=(ind2[(i+1):(i+borne)]-ind2[1:borne])*db2
    fBm[i]=sum(I1)+sum(I2)
  }
  fBm=fBm*n^(-H)*CH
  fBm[1]=0

  time=(0:(n-1))/n
```

```
  param=paste(c("H=", H), collapse=" ")
  param1=paste(c("N=", n, ","), collapse=" ")
  param2=paste(c("Metodas: Stochastinio integralo
  aproksimavimas,",param1, param),
               collapse=" ")
  plot(time, fBm,type='l', main="Trupmeninis Brauno judesys",
  sub=param2,
       xlab="", ylab="")
  drop(fBm)
}
```

- Tikrinių reikšmių metodas

```
fBmEigen=function(H,n){

  imatrix=matrix((1:n)/n,nrow=n,ncol=n)
  jmatrix=matrix((1:n)/n,nrow=n, ncol=n, byrow=TRUE)
  h=H*2
  gamma=0.5*(imatrix^h+jmatrix^h-(abs(imatrix-jmatrix)^h))

  v=rnorm(n)
  gamma.eig=eigen(gamma)
  fbm=gamma.eig$vectors %*% diag(sqrt(gamma.eig$values)) %*%
    solve(gamma.eig$vectors)%*%v
  param=paste(c("H=", H), collapse=" ")
  param1=paste(c("N=", n, ","), collapse=" ")
  time <- (0:(n - 1))/n
  param2=paste(c("Metodas: Tikrinių reikšmių,",param1, param),
               collapse=" ")

  plot(time,fbm,type='l', main="Trupmeninis Brauno judesys",
  sub=param2,
       xlab="", ylab="")
  drop(fbm)
}
```

- Cholesky metodas

```
fBmCholesky=function(H,n){
  if(missing(H))   H=0.5
  if(missing(n))   n=200
  imatrix=matrix((1:n)/n,nrow=n,ncol=n)
  jmatrix=matrix((1:n)/n,nrow=n, ncol=n, byrow=TRUE)
  h=H*2
```

```
  gamma=0.5*(imatrix^h+jmatrix^h-(abs(imatrix-jmatrix)^h))

  v=rnorm(n)
  L=t(chol(gamma))
  fbm=as.vector(L%*%v)
  param=paste(c("H=", H), collapse=" ")
  param1=paste(c("N=", n, ","), collapse=" ")
  time <- (0:(n - 1))/n
  param2=paste(c("Metodas: Cholesky,",param1, param),
  collapse=" ")
  plot(time,fbm ,type='l', main="Trupmeninis Brauno judesys",
  sub=param2,
       xlab="", ylab="")
  drop(fbm)
}
```

- Karhunen – Loeve generavimo metodas

```
fBmKL=function(H,n){
  ksi=rnorm(n)

  h=H*2
  gamma=0.5*(matrix((1:n)/n,nrow=n,ncol=n)^h+matrix((1:n)/n,
  nrow=n, ncol=n, byrow=TRUE)^h- (abs(matrix((1:n)/n, nrow=n,
  ncol=n)-matrix((1:n)/n, nrow=n, ncol=n, byrow=TRUE))^h))

  gamma.eig=eigen(gamma)
  fi=gamma.eig$vectors
  lambda=gamma.eig$values

  fbm=0
  x=matrix(0,n,n)

  for (i in 1:n){
    x[,i]=sqrt(lambda)*ksi*fi[i,]
  }
  fbm=apply(x,2,sum)

  param=paste(c("H=", H), collapse=" ")
  param1=paste(c("N=", n, ","), collapse=" ")
  param2=paste(c("Metodas: KL metodas",param1, param),
               collapse=" ")
  time <- (0:(n - 1))/n
  plot(time,fbm,type='l', main="Trupmeninis Brauno judesys",
```

```
  sub=param2,
       xlab="", ylab="")
  drop(fbm)
}
```

- Levinsono algoritmas. Remiantis Coeurjolly [6]

```
fBmLevinson=function(H,n){
  if(missing(n)) n = 500
  if(missing(H)) H = 0.7

  k = 0:(n - 1)
  H2 = 2 * H
  r = (abs((k - 1)/n)^H2 - 2 * (k/n)^H2 + ((k + 1)/n)^H2)/2

  y = rnorm(n)
  fGn = rep(0, n)
  v1 = r
  v2 = c(0, r[c(2:n)], 0)
  k = - v2[2]
  aa = sqrt(r[1])

  for(j in (2:n)) {
    aa = aa * sqrt(1 - k * k)
    v = k * v2[c(j:n)] + v1[c((j - 1):(n - 1))]
    v2[c(j:n)] = v2[c(j:n)] + k * v1[c((j - 1):(n - 1))]
    v1[c(j:n)] = v
    bb = y[j]/aa
    fGn[c(j:n)] = fGn[c(j:n)] + bb * v1[c(j:n)]
    k = - v2[j + 1]/(aa * aa)
  }
  fBm = cumsum(fGn)
  fBm[1] = 0

  param=paste(c("H=", H), collapse=" ")
  param1=paste(c("N=", n, ","), collapse=" ")
  param2=paste(c("Metodas: Levinson metodas",param1, param),
               collapse=" ")
  time = (0:(n - 1))/n
  plot(time,fBm,type='l', main="Trupmeninis Brauno judesys",
  sub=param2,
       xlab="", ylab="")
  drop(fBm)
}
```

- Bangelių metodas. Remiantis Coeurjolly [6]

```
fBmWave = function(H,n, J)
{
  if(missing(n)) n = 500
  if(missing(H)) H = 0.6
  if(missing(J)) J = 7


  Db20 = c(0.026670057901, 0.188176800078)
  Db20 = c(Db20, 0.52720118932, 0.688459039454)
  Db20 = c(Db20, 0.281172343661, -0.249846424327)
  Db20 = c(Db20, -0.195946274377, 0.127369340336)
  Db20 = c(Db20, 0.093057364604, -0.071394147166)
  Db20 = c(Db20, -0.029457536822, 0.033212674059)
  Db20 = c(Db20, 0.003606553567, -0.010733175483)
  Db20 = c(Db20, 0.001395351747, 0.001992405295)
  Db20 = c(Db20, -0.000685856695, -0.000116466855)
  Db20 = c(Db20, 9.358867e-05, -1.3264203e-05)
  secu <- 2 * length(Db20)


  Db20qmf = (-1)^(0:19) * Db20
  Db20qmf = Db20qmf[20:1]
  nqmf = -18

  prec = 0.006
  hmoy = c(1, 1)
  s = H + 1/2
  d = H - 1/2
  if(H == 1/2) {
    ckbeta = c(1, 0)
    ckalpha = c(1, 0)
  }
  else {
    ckalpha = 1
    ckbeta = 1
    ka = 2
    kb = 2
    while(abs(ckalpha[ka - 1]) > prec) {
      g = gamma(1 + d)/gamma(ka)/gamma(d + 2 - ka)
      if(is.na(g)) g = 0
      ckalpha = c(ckalpha, g)
      ka = ka + 1
```

```
  }


  while(abs(ckbeta[kb - 1]) > prec) {
    g = gamma(kb - 1 + d)/gamma(kb)/gamma(d)
    if(is.na(g)) g = 0
    ckbeta = c(ckbeta, g)
    kb = kb + 1
  }
}
nbmax = max(length(ckbeta), length(ckalpha))
nb0 = n/(2^(J)) + 2 * secu

fs1 = convol(ckalpha, Db20)
fs1 = convol(fs1, hmoy)
fs1 = 2^( - s) * fs1
fs1 = fs1 * sqrt(2)

gs12 = convol(ckbeta, Db20qmf)
gs1 = cumsum(gs12)
gs1 = 2^(s) * gs1
gs1 = gs1 * sqrt(2)

bh=0
nb1 = nb0 + nbmax
bb = rnorm(nb1)
b1 = convol(bb, ckbeta)
bh = cumsum(b1)
bh = bh[c(nbmax:(nb0 + nbmax - 1))]
appro = bh
tappro = length(appro)

dilatation = function(vect)
{
  ldil = 2 * length(vect) - 1
  dil = rep(0, ldil)
  dil[seq(1, ldil, by = 2)] = vect
  drop(dil)
}

for(j in 0:(J - 1)) {
  appro = dilatation(appro)
  appro = convol(appro, fs1)
```

```
    appro = appro[1:(2 * tappro)]
    detail = rnorm(tappro) * 2^(j/2) * 4^( - s) * 2^( - j * s)
    detail = dilatation(detail)
    detail = convol(detail, gs1)
    detail = detail[( - nqmf + 1):( - nqmf + 2 * tappro)]
    appro = appro + detail
    tappro = length(appro)
  }
  debut = (tappro - n)/2
  fBm = appro[c((debut + 1):(debut + n))]
  fBm = fBm - fBm[1]
  fGn = c(fBm[1], diff(fBm))
  fGn = fGn * 2^(J * H) * n^( - H)
  fBm = cumsum(fGn)
  fBm[1] = 0

  param=paste(c("H=", H), collapse=" ")
  param1=paste(c("N=", n, ","), collapse=" ")
  param2=paste(c("Metodas: Bangelių metodas",param1, param),
               collapse=" ")
  time = (0:(n - 1))/n
  plot(time,fBm,type='l', main="Trupmeninis Brauno judesys",
  sub=param2,
       xlab="", ylab="")
    drop(fBm)
}

convol = function(x, y)
{
  if(missing(x) | missing(y)) break
  else {
    a = c(x, rep(0, (length(y) - 1)))
    b = c(y, rep(0, (length(x) - 1)))
    a = fft(a, inverse = F)
    b = fft(b, inverse = F)
    conv = a * b
    conv = Re(fft(conv, inverse = T))
    conv = conv/length(conv)
    drop(conv)
  }
}
```

- Wood – Chano algoritmas. Remiantis Coeurjolly [6]

```
fBmCirc = function(H, n){

  if(missing(n)) n = 100
  if(missing(H)) H = 0.5

  lineC = function(H, n, m){
    k = 0:(m-1)
    H2 = 2*H
    v = (abs((k-1)/n)^H2-2*(k/n)^H2+((k+1)/n)^H2)/2
    ind = c(0:(m/2-1), m/2, (m/2-1):1)
    v = v[ind+1]
    drop(v)
  }

  m = 2
  repeat {
    m = 2*m
    if(m >= (n-1))
      break
  }
  stockm = m

  repeat {
    m = 2*m
    eigenvalC = lineC(H, n, m)
    eigenvalC = fft(c(eigenvalC), inverse = F)
    if((all(Re(eigenvalC) > 0)) | (m > 2^17))
      break
  }

  ar = rnorm(m/2 + 1)
  ai = rnorm(m/2 + 1)
  ar[1] = sqrt(2) * ar[1]
  ar[(m/2 + 1)] = sqrt(2) * ar[(m/2 + 1)]
  ai[1] = 0
  ai[(m/2 + 1)] = 0
  ar = c(ar[c(1:(m/2 + 1))], ar[c((m/2):2)])
  aic = - ai
  ai = c(ai[c(1:(m/2 + 1))], aic[c((m/2):2)])
  W = complex(real = ar, imaginary = ai)

  W = (sqrt(eigenvalC)) * W
  fGn = fft(W, inverse = F)
```

```
  fGn = (1/(sqrt(2 * m))) * fGn
  fGn = Re(fGn[c(1:n)])
  fBm = cumsum(fGn)
  fBm[1] = 0

  param=paste(c("H=", H), collapse=" ")
  param1=paste(c("N=", n, ","), collapse=" ")
  param2=paste(c("Metodas: Wood-Chan metodas",param1, param),
               collapse=" ")
  time = (0:(n - 1))/n
  plot(time,fBm,type='l', main="Trupmeninis Brauno judesys",
  sub=param2,
       xlab="", ylab="")
  drop(fBm)
}
```

H vertinimo algoritmai

- Breton – Coeurjolly metodas

```
est_Breton=function(fBm, conf, alpha){
  n=length(fBm)
  l=2
  if(missing(alpha)) alpha = 0.05
  if (missing(conf)) conf =1 #Jei 0, tai intervalai neskaičiuojami
  ka=2.667
  Sn=0
  for (i in 1:(n-2)){
    Sn[i]=(fBm[i+2]-2*fBm[i+1]+fBm[i])^2
  }
  Sna=sum(Sn)/(n-l)

  #Atvirkštinė funkcija#
  inverse = function (f, lower = -1000, upper = 1000) {
    function (y) uniroot((function (x) f(x) - y), lower = lower,
                         upper = upper)[1]
  }
  g_inverse = inverse(function (x) 2*x*log(n)-log(4-4^x),
  -100000000, 0.999999999)

  xlnl=1-sqrt(2*ka*log(1/(alpha/2)))/sqrt(n-l)
  xrnl=1+sqrt(2*ka*log(1/(alpha/2)))/sqrt(n-l)
```

```
  Hinf=max(0, as.numeric(g_inverse(log(xlnl)-log(Sna))))
  Hsup=min(1, as.numeric(g_inverse(log(xrnl)-log(Sna))))
  Hest=as.numeric(g_inverse(-log(Sna)))
  if (conf == 1)
    (return(c(Hinf,Hsup)))
  else (return(c(Hest)))
}
```

- Salomono (Karhunen – Loeve) vertinimo algoritmas

```
est_salomon=function(fBm,q){
  if(missing(q)) q = 1
  #q nurodo, kas kelintą elementą reikia naudoti renkant imtį
  N=length(fBm)
  Yt=matrix(0,N/q,94)
  teta1=0.05
  for (l in 1:94) {
   teta=l*0.01+teta1

    imatrix=matrix((1:N)/N,nrow=N,ncol=N)
    jmatrix=matrix((1:N)/N,nrow=N, ncol=N, byrow=TRUE)
    gamma=0.5*(imatrix^(2*teta)+jmatrix^(2*teta)-
    (abs(imatrix-jmatrix)^(2*teta)))
    drop(imatrix)
    drop(jmatrix)

    gamma.eig=eigen(gamma)
    fi=gamma.eig$vectors
    lambda=gamma.eig$values

    sigma=matrix(0,N/q,N/q)
    for (j in 1:N/q){
      for (i in 1:N/q){
        sigma[i,j]=sum(lambda*fi[i*q,]*fi[j*q,])
      }
    }

    Xt=fBm[which((1:N)%%q==0, arr.ind=TRUE)]

    sig.eig=eigen(sigma)
    Yt[,l]=solve(sig.eig$vectors %*% diag(sqrt(sig.eig$values))
                 %*% solve(sig.eig$vectors))%*%Xt
  }
  z=abs(apply((Yt^2),2,sum)/(N/q)-1)
```

```
  min(z)
  teta_est=match(min(z),z)*0.01+teta1
  return(teta_est)
}
```

- Logaritminės periodogramos metodas. Remiantis Coeurjolly [6]

```
est_LogPer = function(fBm)
{
  m1 = 1
  m2 = trunc((length(fBm) - 1)/2)
  fGn = c(fBm[1], diff(fBm))
  n = length(fGn)

  I.lambda = (Mod(fft(fGn, inverse = F)))^2/(2*pi*n)
  I.lambda = I.lambda[m1:m2]
  lambda = (2 * pi * (m1:m2))/n

  Reg = lsfit(log(lambda), log(I.lambda), intercept = T)
  Hest = 0.5 * (1 - Reg$coef[2])
  drop(Hest)
}
```

- Lobato metodas. Remiantis Coeurjolly [6]

```
est_Lobato <- function(fBm)
{
  q = 0.27
  m1 = 1
  m2 = trunc((length(fBm) - 1)/2)
  fGn = c(fBm[1], diff(fBm))
  n = length(fGn)

  I.lambda = (Mod(fft(fGn, inverse = F)))^2/(2*pi*n)
  Iqm = I.lambda[m1:(trunc(q * m2))]
  Im = I.lambda[m1:m2]
  Hest = 1 - log(sum(Iqm)/sum(Im))/(2 * log(q))
  drop(Hest)
}
```

- Diskrečios variacijos metodas (1). Remiantis D.Melichovu [10]

```
est_fH1 = function(X) {
```

```
  X2n = X
  dX2n = X2n[-1] - X2n[-length(X2n)]
  Xn = X2n[-c(seq(2,length(X2n),2))]
  dXn = Xn[-1] - Xn[-length(Xn)]
  V2n = sum(dX2n^2)
  Vn = sum(dXn^2)
  H = 1/2 - 1/(2*log(2)) * log(V2n/Vn)
  return(H)
}
```

- Diskrečios variacijos metodas (2). Remiantis D.Melichovu [10]

```
est_fH2 = function(X) {
  X2n = X
  dX2n = X2n[-1] - X2n[-length(X2n)]
  d2X2n = dX2n[-1] - dX2n[-length(dX2n)]
  Xn = X2n[-c(seq(2,length(X2n),2))]
  dXn = Xn[-1] - Xn[-length(Xn)]
  d2Xn = dXn[-1] - dXn[-length(dXn)]
  V2n = sum(d2X2n^2)
  Vn = sum(d2Xn^2)
  H = 1/2 - 1/(2*log(2)) * log(V2n/Vn)
  return(H)
}
```

- Mažiausių kvadratų metodas. Remiantis Coeurjolly [6]

```
VaPkolST = function(fBm, k, a, M)
{
  if(missing(fBm)) fBm = circFBM(n = 500, H = 0.6)
  if(missing(k)) k = 2
  if(missing(a))
    a = c(0.4829629, -0.8365163, 0.22414386, 0.12940952)
  if(missing(M)) M = 5
  N = length(fBm)

  dilatation = function(a, m)
  {
    la = length(a)
    am = rep(0, m * la - 1)
    am[seq(1, m * la - 1, by = m)] = a
    drop(am)
  }
```

```
  la = length(a)
  Vam = rep(0, M)
  SNkam = rep(0, M)
  Vam = filter(fBm, a, sides = 1)
  Vam = Vam[ - (1:(la - 1))]
  SNkam[1] = mean(abs(Vam)^k)
  for(m in (2:M)) {
    am = dilatation(a, m)
    Vam = filter(fBm, am, sides = 1)
    lam = m * la - 1
    Vam = Vam[ - (1:(lam - 1))]
    SNkam[m] = mean(abs(Vam)^k)
  }
  LN = log(SNkam)
  m = 1:M
  Reg = lsfit(log(m), LN, intercept = T)
  Hols = Reg$coef[2]/k

  drop(Hols)
}

piaH = function(a, H, i)
{
  if(missing(a)) a = c(1, -1)
  if(missing(H)) H = 0.6
  if(missing(i)) i = 0
  l = length(a) - 1
  d = l + 1
  mat = matrix(rep(0, d * d), ncol = d)
  for(q in (0:l)) {
    for(r in (0:l)) {
      z = a[q + 1] * a[r + 1] * abs(q - r + i)^(2 * H)
      mat[q + 1, r + 1] = -0.5 * z
    }
  }
  piaH.i = sum(apply(mat, 1, sum))
  drop(piaH.i)
}
```

Literatūra

[1] ABRY, P.; SELLAN, F. T. The Wavelet – Based Synthesis for Fractional Brownian Motion Proposed by F.Sellan and Y.Meyer: Remarks and Fast Implementation. Applied and computational harmonic analysis, 3: 377–383. 1996.

[2] BRETON, J. C.; COEURJOLLY, J.F. Confidence intervals of the Hurst parameter of a fractional Brownian motion based on finite sample size. Stat Inference Stoch Process, Springer. 2011.

[3] BRETON, J. C.; NOURDIN, I.; PECCATI, G. Exact confidence intervals for the Hurst parameter of a fractional Brownian motion. Electronic Journal of Statistics, 3: 416–425. 2009.

[4] DIEKER, A. B.; MANDJES, M. On spectral simulation of fractional Brownian motion. 2003.

[5] DIEKER, A. B. Simulation of fractional Brownian motion. 2004.

[6] COEURJOLLY, J. F. Simulation and indentification of the fractional Brownian motion: a bibliographical and comparative study. 2000.

[7] KRUOPIS, J. *Matematinė statistika*: vadovėlis, 2–asis leidimas, Vilnius: Mokslo ir enciklopedijų leidykla. 1993. ISBN 5–420–01015–1.

[8] KUBILIUS, K. *Atsitiktinių procesų statistika*: konspektas. 2012.

[9] MANDELBROT, B.; VAN NESS, J.W. Fractional brownian motions, fractional noises and applications. 1968.

[10] MELICHOV, D., Apie stochastinių diferencialinių lygčių sprendinių Hursto indekso vertinimą, Vilnius: Technika. 2011.

[11] SALOMON, L. A.; FORT, J–C. Estimation of the Hurst parameter in some fractional processes. Journal of Statistical Computation and Simulation, 3: 542–554. 2013.