KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

MATEMATINĖS SISTEMOTYROS KATEDRA

Mantas Landauskas

MARKOVO GRANDINIŲ MONTE KARLO METODO TAIKYMAS STOCHASTINĖMS SISTEMOMS MODELIUOTI

Magistro darbas

Vadovas
doc. dr. E. Valakevičius

KAUNAS, 2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

MATEMATINĖS SISTEMOTYROS KATEDRA

TVIRTINU
Katedros vedėjas
prof. habil.dr. V.Pekarskas
2011 06 02

MARKOVO GRANDINIŲ MONTE KARLO METODO TAIKYMAS STOCHASTINĖMS SISTEMOMS MODELIUOTI

Taikomosios matematikos magistro baigiamasis darbas

Vadovas
( ) doc. dr. E. Valakevičius
2010 06 01

Recenzentas
( ) doc. dr. V. Pilkauskas
2011 05 30

Atliko
FMMM 9 gr. stud.
( ) M. Landauskas
2010 05 30

KAUNAS, 2011

KVALIFIKACINĖ KOMISIJA

Pirmininkas:	Leonas Saulis, profesorius (VGTU)
Sekretorius:	Eimutis Valakevičius, docentas (KTU)
Nariai:	Algimantas Jonas Aksomaitis, profesorius (KTU)
	Vytautas Janilionis, docentas (KTU)
	Vidmantas Povilas Pekarskas, profesorius (KTU)
	Rimantas Rudzkis, habil. dr., vyriausiasis analitikas (DnB NORD Bankas)
	Zenonas Navickas, profesorius (KTU)
	Arūnas Barauskas, dr., vice-prezidentas projektams (UAB „Baltic Amadeus“)

Landauskas M. The application of Markov chain Monte Carlo method to modelling of stochastic systems : Master's work in applied mathematics / supervisor dr. assoc. prof. E. Valakevičius; Department of Mathematical Research in Systems, Faculty of Fundamental Sciences, Kaunas University of Technology. – Kaunas, 2011. – 107 p.

SUMMARY

The application of Markov chain Monte Carlo (MCMC) technique for two stochastic systems was investigated in this paper. A simple single channel system with a queue was modelled considering the service time distribution is of unknown form. Analysis showed that MCMC is possible to apply and gives adequate results.

The custom approach of constructing a proposal density for MCMC was proposed and applied to find the value of a European call option as the classical techniques can not be applied in all the cases. This custom approach leads the enhanced MCMC to be applied to any empirical data having no assumptions about its probability density.

Both applications were programmed in C++. This useful tool can be used for pricing the European option of a real stock having its historical prices. Modelling a theoretical queuing system with the distribution of the service time of an unknown form is also implemented. A MATLAB approach for modelling the dynamics of a stock prices with enhanced MCMC can be found in appendix 2 and 3.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

ĮVADAS

Darbo tikslas yra panaudoti Markovo grandinių Monte Karlo (MCMC) metodą dviem stochastinėms sistemoms modeliuoti. Viena jų – vienkanalė sistema su eile, kurios dydis nėra apribotas, kita – akcijos kainos dinamika.

MCMC metodu generuojamų atsitiktinių dydžių (a.d.) seka yra Markovo grandinė. Tyrimo tikslas buvo išsiaiškinti kaip šis faktas veiks paprasčiausios vienkanalės aptarnavimo sistemos modeliavimo šiuo metodu rezultatus. Nustatyta, kad verta metodą taikyti aptarnavimo sistemai, tačiau prieš tai svarbu sumaišyti MCMC metodu generuojamų a.d. aibę, kad jie būtų nepriklausomi.

Modeliuojant tam tikros akcijos kainos dinamiką pagrindinė problema yra akcijos grąžos pasiskirstymas. Sukonstravus universalų akcijų kainų dinamikos modelį, buvo surastas bazinio aktyvo pasiskirstymas sandorio pabaigoje ir įkainotas europietiškas pasirinkimo sandoris.

Pasirinkimo sandoris yra išvestinė finansinė priemonė (*derivative security*), t.y. toks vertybinis popierius (VP), kurio vertė priklauso nuo kitų vertybinių popierių, vadinamų bazinėmis finansinėmis priemonėmis (*underlying assets*). Baziniai finansiniai aktyvai skirstomi į 5 grupes: akcijos, valiuta, palūkanų normos, akcijų rinkos indeksai, prekės. O išvestinės finansinės priemonės skirstomos į 3 pagrindines grupes: pasirinkimo sandoriai (*options*), ateities ir būsimieji sandoriai (*futures and forwards*), apsikeitimo sandoriai (*swaps*). Aktualiausia yra įkainoti pasirinkimo sandorius, nes sudėtingesni sandoriai gali būti išreiškiami pasirinkimo ir ateities sandoriais. Pasirinkimo bei ateities sandoriai yra laikomi pagrindinėmis išvestinėmis finansinėmis priemonėmis.

Pasirinkimo sandoris gali būti panaudotas apsidrausti nuo rizikos, susijusios su kainos nepastovumu arba spekuliavimo VP biržoje tikslais. Sandorio pardavėjas laukia ir, sandorio turėtojui pareikalavus, privalo parduoti sutartą prekę arba aktyvą sandorio įvykdymo momentu, todėl kaip kompensaciją gauna tam tikrą sumą (premiją) sandorio pasirašymo metu. Ši suma toliau vadinama sandorio kaina.

Nustatyti tikslią sandorio kainą galima tik tada, kai žinoma, kokia bazinių VP kaina bus ateityje. Įkainojimas atliekamas panaudojant įvairius akcijų kainų dinamikos modelius, nes tiksliai nežinoma, kokia bus akcijos kaina ateityje. Kiekviename iš tokių modelių kainos kitimas turi būti rizikai neutralus. Neutralumas rizikai gali būti paaiškintas vienodu investuotojo požiūriu į rizikingo aktyvo grąžą ir į tokią pačią nerizikingo aktyvo grąžą. Darbe aprašomi 2 dažniausiai naudojami metodai pasirinkimo sandoriams įkainoti: Black-Scholes formulė ir Monte Karlo modeliavimas. Abu metodai paremti Brauno judesio procesu.

1. TEORINĖ DALIS

Šioje dalyje aprašomas Markovo grandinių Monte Karlo (MCMC) metodas ir dvi pasirinktos juo modeliuojamos stochastinės sistemos: vienkanalė aptarnavimo sistema ir akcijos kainos dinamika. Pastarąja remiantis nagrinėtas pasirinkimo sandorio įkainojimas. Taip pat aptarti pasirinkimo sandoriai ir prielaidos, kurios būtinai turi būti išpildytos, kad nagrinėjami modeliai būtų adekvatūs. Aptariamos sistemos su eile charakteristikos bei jų apskaičiavimas.

Darbe nagrinėjami europietiškieji pasirinkimo sandoriai, kurių tiksli kaina apskaičiuojama pagal Black-Scholes formulę. Tačiau pateikta metodika gali būti panaudota įkainoti sudėtingiems sandoriams (pavyzdžiui azijietiškiems), kurių vertė randama tik naudojantis Monte Karlo modeliavimu. Dauguma rezultatų pateikta pasirinkimo pirkti sandoriams, tačiau darbo metu sukurta programa skaičiuoja ir pasirinkimo parduoti sandorių kainas.

1.1. MARKOVO GRANDINIŲ MONTE KARLO METODAS

1.1.1. MCMC METODO SAMPRATA

Imkime tikimybinį skirstinį $\pi(x)$, $x \in \mathbf{R}^n$. Tarkime, kad mūsų uždavinys yra ištirti šį skirstinį arba rasti kokį nors įvertį $\hat{\theta}(x)$, pavyzdžiui $E_\pi(f(x))$, čia $f(x)$ yra kokia nors kintamojo x funkcija. Šiam tikslui pasiekti reikia mokėti generuoti $x_i \sim \pi(x)$. Monte Karlo metodu galima generuoti $x_1, x_2, ..., x_n \sim \pi(x)$ ir paskaičiuoti vidurkį arba kitą ieškomą įvertį naudojant integralą.

$$\theta = E_\pi(f(x)) = \int f(x)\pi(x)dx, \tag{1.1.1}$$

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} f(x_i). \tag{1.1.2}$$

Tokiu būdu apskaičiuotas įvertis yra nepaslinktas ir suderintas.

$$E(\hat{\theta}) = \frac{1}{n}\sum_{i=1}^{n} E(f(x_i)) = \frac{1}{n}\sum_{i=1}^{n} \theta = \theta, \tag{1.1.3}$$

$$Var(\hat{\theta}) = \frac{1}{n^2}\sum_{i=1}^{n} Var(f(x_i)) = \frac{Var(f(x))}{n},\ \sigma(\hat{\theta}) = \frac{\sigma(f(x))}{\sqrt{n}}. \tag{1.1.4}$$

$$\lim_{n\to\infty} \sigma(\hat{\theta}) \to 0. \tag{1.1.5}$$

Uždavinio sprendimą nusako aukščiau pateiktos formulės, bet kyla klausimas, kaip generuoti $x_i \sim \pi(x)$, jeigu nežinoma kaip tai padaryti. Nesvarbu, kad turimas $\pi(x)$, bet galbūt nežinoma $\pi^{-1}(x)$ analizinė išraiška. Ši tankio funkcija taip pat gali turėti per daug sudėtingą analizinę išraišką, kad galima būtų taikyti grafinį atsitiktinių skaičių generavimo metodą arba išreikšti ją žinomais pasiskirstymais. Tačiau reikiamus atsitiktinius dydžius galima gauti panaudojus Markovo grandinę. Naudojamos Markovo grandinės stacionarusis skirstinys turi būti lygus $\pi(x)$, kad ji būtų tinkama

modeliavimui [8, 12]. MCMC metodo tikslas ir yra sukonstruoti tokią Markovo grandinę $\{X_i\}_{i=0}^{\infty}$ ir tokiu būdu, kad $\lim_{i\to\infty} P(X_i = x) = \pi(x)$.

Modeliuojant atsitiktinį procesą nuo pradinės būsenos $x_0 \sim f(x_0)$, pastarąją galima, to nenorint, pasirinkti tokią, kad dar po kelių grandinės žingsnių generuojamų atsitiktinių dydžių seka nutols nuo jos. Tai nebūtinai reiškia, kad pagal Markovo grandinę generuojami atsitiktiniai dydžiai konverguoja, nes grandinėje pastoviai vyksta perėjimai tarp būsenų. Nepaisant to, $x_1, x_2, ..., x_n$ nekonvergavimas yra būtinas $P(X_i = x)$ konvergavimui. Dėl to modeliuojant grandinė turi „prarasti atmintį“, t.y. į atsitiktinių skaičių imtį imami tokie x_i, kad $x_m, x_{m+1}, ... \sim \pi(x)$.

Markovo grandinę galima nusakyti pradine būsena arba pradiniu pasiskirstymu $P(X_0 = x_0)$ bei perėjimo tarp būsenų branduoliu (perėjimų tikimybių matrica) $P(y|x) = P(X_{i+1} = y | X_i = x)$.

Stacionarusis pasiskirstymas $\pi(x) = \lim_{i\to\infty} f(x_i)$ yra vienintelis, jeigu Markovo grandinė yra nesuprastinama, neperiodinė ir kiekvieną jos būseną galima pasiekti iš kitos būsenos per baigtinį skaičių žingsnių. Tokia grandinė yra vadinama ergodine Markovo grandine. Grandinės neperiodiškumui pakanka, kad $\exists\ x \in \Omega$, toks, kad $P(x, x) > 0$. Taigi kai grandinė yra ergodinė, turime:

$$\pi(y) = \sum_{x\in\Omega} \pi(x) P(y|x),\ \forall y \in \Omega. \tag{1.1.6}$$

Markovo grandinė yra vadinama apgręžiama, jeigu $\pi(x)P(y|x) = \pi(y)P(x|y),\ \forall x, y \in \Omega$.

$$\sum_{x\in\Omega} \pi(x)P(y|x) = \sum_{x\in\Omega} \pi(y)P(x|y) = \pi(y)\sum_{x\in\Omega} P(x|y) = \pi(y). \tag{1.1.7}$$

Po sumos ženklu esantį reiškinį galima traktuoti kaip vieną iš tiesinių lygčių sistemos lygčių. Todėl (1.1.6) užrašome kaip lygčių sistemą:

$$\begin{cases} \pi(x_2) = \pi(x_1)P(x_2|x_1) + \pi(x_2)P(x_2|x_2) + ... + \pi(x_n)P(x_2|x_n); \\ ... \\ \pi(x_n) = \pi(x_1)P(x_n|x_1) + \pi(x_2)P(x_n|x_2) + ... + \pi(x_n)P(x_n|x_n); \end{cases} \tag{1.1.8}$$

čia $n := |\Omega|$. Lygčių yra $(n-1)$, o nežinomųjų $n(n-1)$. Iš čia seka, kad $\exists$ be galo daug perėjimo tarp būsenų branduolių, tokių, kad jais nusakomų Markovo grandinių stacionarusis pasiskirstymas būtų $\pi(x)$. Jeigu pareikalausime Markovo grandinės apgręžiamumo, tai nežinomųjų bus tiek pat, o lygčių $n(n-1)/2$. Pradinės problemos sprendimui (x_i generavimui) užtenka tik vieno iš šių branduolių.

1.1.2. METROPOLIO-HASTINGSO ALGORITMAS

Norint sukonstruoti perėjimo tarp būsenų branduolį, reikia pasirinkti kitą (alternatyvų) branduolį ir dauginti jį iš būsenos pasikeitimo tikimybės. Ši tikimybė, aišku, priklausys nuo pasirinkto branduolio. Tokia ir yra šio metodo esmė [2, 4, 5].

$$P(y|x) = Q(y|x)\alpha(y|x),\ y \neq x,\ \alpha(y|x) \in [0;1]. \tag{1.1.9}$$

Kadangi konstruojama Markovo grandinė, tai jos perėjimų tikimybių matrica bus stochastinė ir turėsime:

$$\sum_{y\in\Omega} P(y|x) = 1,\ \forall x \in \Omega. \tag{1.1.10}$$

$$P(x|x) = 1 - \sum_{x\neq y} Q(y|x)\alpha(y|x). \tag{1.1.11}$$

Pasinaudojus Markovo grandinės apgręžiamumu:

$$\pi(x)Q(y|x)\alpha(y|x) = \pi(y)Q(x|y)\alpha(x|y),\ \forall x \neq y. \tag{1.1.12}$$

Perėjimo tarp būsenų branduolys $Q(y|x)$ yra pasirinktas, todėl nežinoma lieka tik būsenos pasikeitimo tikimybė α. Bendrasis (1.1.12) lygties sprendinys yra:

$$\alpha(y|x) = r(x,y)\pi(y)Q(x|y),\ r(x,y) = r(y,x). \tag{1.1.13}$$

$$\begin{cases} \alpha(y|x) = r(x,y)\pi(y)Q(x|y) \le 1 \Rightarrow r(x,y) \le \dfrac{1}{\pi(y)Q(x|y)}; \\ \alpha(x|y) = r(x,y)\pi(x)Q(y|x) \le 1 \Rightarrow r(x,y) \le \dfrac{1}{\pi(x)Q(y|x)}; \end{cases} \tag{1.1.14}$$

čia $r(x,y)$ reikia parinkti taip, kad nagrinėjama metodika būtų kuo efektyvesnė. Viena iš taktikų pasirenkant $r(x,y)$ yra ta, kad norime kuo didesnės $\alpha(y|x)$, kad grandinėje vyktų kuo daugiau perėjimų ir taip būtų sumodeliuota kuo daugiau reikšmių iš Ω.

$$\alpha(y|x) = \min\left(1, \frac{\pi(y)Q(x|y)}{\pi(x)Q(y|x)}\right). \tag{1.1.15}$$

Iš (1.1.15) matyti, kad $\pi(x)$ gali būti žinomas konstantos tikslumu $\pi(x) = c \cdot h(x)$. Tai dažnai užrašoma kaip $\pi(x) \propto h(x)$. Arba iš kitos pusės: mūsų pasirenkamas pasiskirstymas skiriasi nuo duotojo multiplikatyviosios konstantos tikslumu, o tai reiškia, kad grafiškai pavaizduotas yra panašios formos.

Taikant aptartą $P(y|x)$ konstravimo principą, gaunama imtis $x_1, x_2, ..., x_n \sim \pi(x)$.

1.2. APTARNAVIMO SISTEMOS

1.2.1. PAPRASČIAUSIA VIENKANALĖ SISTEMA *M/G/1* SU EILE

Nagrinėkime paprasčiausią vienkanalę aptarnavimo sistemą be apropojimo eilės dydžiui. Tokia sistema žymima santrumpa *M/G/1*. Pirmasis žymuo nusako paraiškų atėjimo srauto tipą (M – paprasčiausias, t.y. puasoninis), antrasis – paraiškų aptarnavimo srauto tipą (G – bendrojo tipo, agl. „general“), trečiasis žymi kanalų skaičių. Pabrėžiant, kad eilė yra neribojama, sistema dar žymima $M/G/1/\infty$. Sakykime, kad į sistemą ateina paprasčiausias paraiškų srautas su intensyvumu λ, o aptarnavimo laikas eksponentinis su intensyvumu μ. Tokiu atveju stacionariosios sistemos būsenų tikimybės egzistuoja tik tada, kai sistemos apkrova $\rho = \frac{\lambda}{\mu} < 1$, priešingu atveju eilė neaprėžtai didėja. Sistemos būsenos apibrėžtos 1.2.1 lentelėje.

1.2.1 lentelė

Sistemos su eile būsenų apibrėžimas

Žymėjimas	Būsenos apibūdinimas
S_0	aptarnavimo sistema laisva
S_1	1 paraiška aptarnaujama, bet eilės nėra
S_2	1 paraiška aptarnaujama ir 1 paraiška stovi eilėje
...	...
S_k	1 paraiška aptarnaujama ir $k-1$ paraiškų stovi eilėje

Aprašytos sistemos su eile stacionarios būsenų tikimybės yra $p_0 = 1-\rho$, $p_k = \rho^k(1-\rho)$, $k = 1; 2; \ldots$. Sistemos darbo efektyvumo charakteristikos aprašytos 1.2.2 lentelėje.

1.2.2 lentelė

Sistemos su eile charakteristikos

Charakteristika	Aprašymas
$A = \lambda$	vidutinis aptarnaujamų paraiškų skaičius
$\overline{Z} = \frac{\rho}{1-\rho}$	vidutinis paraiškų skaičius sistemoje
$\overline{r} = \frac{\rho^2}{1-\rho}$	vidutinis paraiškų skaičius eilėje
$\overline{t_{sist}} = \frac{\rho}{\lambda(1-\rho)}$	vidutinis paraiškos buvimo laikas sistemoje
$\overline{t_{eil}} = \frac{\rho^2}{\lambda(1-\rho)}$	vidutinis paraiškos buvimo laikas eilėje
$p_{apt} = \frac{\lambda}{\mu} = \rho$	tikimybė, kad kanalas užimtas

Sistemos *M/G/1* su eile, laiko charakteristikos skaičiuojamos pagal Litlo formules [10]. Vidutinis paraiškų skaičius eilėje ir sistemoje apskaičiuojamas pagal Polečeko-Chinčino formulę [1].

1.2.2. APTARNAVIMO SISTEMOS MODELIAVIMAS

Imkime vienkanalę aptarnavimo sistemą be apribojimo eilės ilgiui. Sakykime, kad momentu t_i ateina i – toji paraiška į sistemą. Ši paraiška eilėje laukia laiką t_i^{eil}, yra aptarnaujama per laiką t_i^{apt} tokiu būdu išbūdama sistemoje laiką $t_i^{sist} = t_i^{eil} + t_i^{apt}$. Dydžiai t_i ir t_i^{apt} yra atsitiktiniai ir generuojami pagal jų atitinkamus tikimybinius pasiskirstymus.

Naujai atėjusi paraiška gali rasti užimtą kanalą. Todėl reikia žinoti laiką, kurį paraiška turi laukti eilėje iki kol bus pradėta aptarnauti. Tarkime, kad žinoma, kiek eilėje laukė (i – 1) – oji paraiška. Pastarajai uždelsus t_{i-1}^{eil}, nesunku pastebėti, kad $t_i^{eil} = \max\left\{0, t_{i-1}^{eil} + t_{i-1} + t_{i-1}^{apt} - t_i\right\}$. Pati pirmoji į sistemą atėjusi paraiška iš karto yra aptarnaujama, todėl $t_1^{eil} = 0$. Tyrimo metu priimta, kad sistema nenutrūkstamai pradeda dirbti tuo pačiu momentu, kai į ją kreipiasi pirmoji paraiška.

Toliau bus pateiktas konkretus pavyzdys, iliustruojantis aptartas formules. Nesigilinant į tai, kokie skirstiniai aprašo sistemos procesus, 1.2.3 lentelėje pateikti paraiškų atėjimo į sistemą momentai ir joms priskirtos atsitiktinės aptarnavimo trukmės.

1.2.3 lentelė

Sistemos charakteristikos

Paraiškos atėjimo momentas, t_i	Paraiškos aptarnavimo trukmė, t_i^{apt}	Paraiškos laikas eilėje, t_i^{eil}
0	1	0
2	3	max{0, 0 + 0 + 1 - 2} = 0
4	2	max{0, 2 + 0 + 3 - 4} = 1
5	4	max{0, 4 + 1 + 2 - 5} = 0
7	2	max{0, 5 + 0 + 4 - 7} = 2
10	1	max{0, 7 + 2 + 2 - 10} = 1
13	2	max{0, 10 + 1 + 1 - 13} = 0

Atliekant modeliavimą ir ieškant eilės charakteristikų, fiksuojami momentai, kai eilė didėja ir kai mažėja. Kai $t_i^{eil} > 0$, tai eilė didėja, o jos padidėjimo momentas yra t_i. Eilės sumažėjimo momentas fiksuojamas taip: $t_i + t_i^{apt} + t_i^{eil}$.

1.4 lentelė

Eilės pokyčiai ir dydžio kitimas

Laiko momentas	0	1	2	3	4	5	6	7	8	9	10	11	12
Eilės pokytis	0	0	0	+1	0	0	+1 –1	0	0	+1	–1	–1	0
Eilės dydis	0	0	0	1	1	1	1	1	1	2	1	0	0

Vidutinis aprašomo pavyzdžio eilės dydis yra $9/13$ žmogaus. Toks įvertis gautas susumavus 1.2.4 lentelės paskutinę eilutę ir padalijus sumą iš laiko momentų skaičiaus.

Paraiškos aptarnavimo laikas nepriklauso nuo to, kiek ji stovėjo eilėje, tačiau jos laikas laukiant eilėje (tuo pačiu ir buvimo sistemoje laikas) priklauso nuo kitų paraiškų aptarnavimo trukmių. Tai

reiškia, kad t_i^{apt} ir t_i^{eil} yra priklausomi atsitiktiniai dydžiai. Todėl vertinant pagrindines sistemos charakteristikas reikia naudotis tokiomis tapatybėmis:

$$Et_i^{sist} = Et_i^{eil} + Et_i^{apt}, \quad \textbf{(1.2.1)}$$

$$Dt_i^{sist} = Dt_i^{eil} + Dt_i^{apt} + 2\operatorname{cov}\left(t_i^{eil}, t_i^{apt}\right). \quad \textbf{(1.2.2)}$$

1.3. PASIRINKIMO PIRKTI IR PARDUOTI SANDORIAI

Pasirinkimo pirkti sandoris (call option) jo savininkui suteikia teisę, bet ne pareigą, ateityje sandorio pabaigoje arba anksčiau pirkti finansinį aktyvą už iš anksto nustatytą kainą, vadinamą ceremonijos (įvykdymo) kaina (exercise/strike price).

Pavyzdys. 6 mėnesių trukmės pasirinkimo pirkti GM akcijas sandoris su 50$ ceremonijos kaina kainuoja 6$. Sakykime sandorio pasirašymo momentu GM akcijų kaina yra 51$. Sandoris yra patrauklus (*in the money*), nes įvykdyti jį pirkimo momentu (vertinant tik pajamas, o ne pelną) apsimoka. Jeigu sandorio pabaigoje GM akcijų kaina nukristų iki 10$, tai investuotojas prarastų tik 6$, nes jis neprivalo įvykdyti sandorio. Tačiau jeigu paskutinę ar kurią nors ankstesnę dieną akcijų kaina pakiltų iki, pavyzdžiui, 70$, tai investuotojas už vieną tokį sandorį uždirbtų $70-50-6=14\,\$$. Procesas vyksta taip: investuotojas įvykdo sandorį, t.y. perka akciją už 50 $ ir iš karto ją rinkoje parduoda už esamą 70$ kainą. Įvertinus sandorio kaštus, lieka 14$ pelnas.

Pasirinkimo parduoti sandoris (put option) jo savininkui suteikia teisę, bet ne pareigą, ateityje sandorio pabaigoje arba anksčiau parduoti finansinį aktyvą už iš anksto nustatytą kainą, vadinamą ceremonijos (įvykdymo) kaina (exercise/strike price).

Pavyzdys. Pasirinkimo parduoti GM akcijas sandoris su 50$ ceremonijos kaina parduotas 2008 metų gruodžio mėnesį už 5,50$. Sandorio pabaiga yra 2009 metų birželio mėnuo. Tarkime akcijos kaina sandorio pabaigoje bus 40$. Sandorį įvykdžius pelnas yra $50-40-5{,}50=4{,}50\,\$$. Šiuo atveju teigiamas pelnas bus gautas, kai sandorio pabaigoje akcijos kaina yra $< 44{,}50\$$.

Pasirinkimo sandoriais prekiaujančioje biržoje sandorių galiojimas baigiasi šeštadienį po 3 to mėnesio, kurį turi baigtis sandoris, penktadienio.

1.3.1. PAGRINDINĖS AKCIJŲ KAINŲ SAVYBĖS IR PRIELAIDOS

Šiuo metu naudojami akcijų kainų dinamikos modeliai remiasi tokiomis pagrindinėmis savybėmis [15]:

1) Akcijos kaina yra atsitiktinė. Žinant kainą šiandien, negalima žinoti rytojaus kainos.
2) Akcijų kainos visada yra griežtai teigiamos. Tai reiškia, kad nenagrinėjamos bankrutavusių firmų akcijos.

3) Priimta prielaida, kad akcijų kainos kinta tolydžiai. Per nykstamai trumpą laiko tarpą kainos pokytis yra nykstamai mažas. Be to, kai stebimas laiko intervalas artėja į 0, tai akcijos kainos pokytis taip pat artėja į nulį.
4) Akcijų vertė turi tendenciją didėti laike. Per ilgą laiko tarpą akcijos kaina nebūtinai padidės, bet vidutinė grąža per tą laiko tarpą bus didesnė nei per trumpesnį laikotarpį.
5) Akcijų vidutinis pelno normos neapibrėžtumas $E(\sigma)$ turi tendenciją didėti su laiku. Jei šiandien žinome akcijos kainą, tai rytoj jos dispersiją bus maža, bet po ilgesnio laiko tarpo (pvz. pusmečio) ji bus žymiai didesnė.

Remiantis šiomis prielaidomis nesunku suvokti, kad dabartiniai akcijų dinamikos modeliai tinka tik augančios rinkos atveju. Esant ekonominiam nuosmukiui negalima pasitikėti esamais akcijų kainų dinamikos modeliais, nes jų prielaidos yra pažeidžiamos. Dėl šios priežasties yra poreikis tobulinti esamus arba kurti naujus, abstraktesnius modelius.

1.3.2. AKCIJŲ ISTORINIŲ KAINŲ ANALIZĖ IR KLASIKINIO MODELIO PARAMETRŲ ĮVERČIAI

Prieš pradedant kalbėti apie pasirinkimo snadorių įkainojimo metodus, reikia aptarti kaip taisyklingai jiems turi būti pateikiami duomenys. Kiekvienas iš metodų naudoja akcijų kainų grąžų logaritmų vidurkį ir standartinį nuokrypį.

Akcijų kainų grąžų logaritmų vidurkį ir standartą randame iš istorinių duomenų. Grąžos logaritmo standartinis nuokrypis (kintamumas) σ yra grąžų nepastovumo matas. Akcijoms σ dažniausiai būna nuo 15% iki 60%. Standartinį nuokrypį galima apibrėžti ir kaip akcijos grąžos vidurkio (skaičiuojamo tolygiosiomis palūkanomis per 1 metus) standartą. Kadangi $\frac{\Delta S}{S} \sim N\left(\mu\Delta t, \sigma\sqrt{\Delta t}\right)$, tai esant mažam Δt dydis $\sigma\sqrt{\Delta t}$ bus apytiksliai lygus kainų grąžų standartui per laikotarpį Δt. Pavyzdžiui, jei akcijos grąžos savaitės kintamumas yra $\sigma = 0{,}2$, tai vienos dienos procentinis kainos pokytis bus $0{,}2 \cdot \sqrt{\frac{1}{7}} = 0{,}0756 = 7{,}56\%$. Matome, kad ateities kainos nepastovumas yra proporcingas $\sqrt{t}$, kai nagrinėjamas laiko periodas *t*.

Skaičiuojant akcijų kainų grąžų kintamumą iš praeities duomenų, akcijos kainos stebimos fiksuotais laiko intervalais. Pažymėkime $u_i = \ln\left(\frac{S_i}{S_{i-1}}\right)$. u_i standartinis kvadratinis nuokrypis apskaičiuojamas pagal 1.3.1 formulę.

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(u_i - \frac{1}{n}\sum_{i=1}^{n}u_i\right)^2}\,. \qquad \textbf{(1.3.1)}$$

Iš anksčiau minėtų pastabų matyti, kad nagrinėjant laikotarpį t (metais), $\sigma(u_i) = \sigma\sqrt{t}$. Tada metinio kintamumo įvertis yra $\hat{\sigma} = \frac{s}{\sqrt{t}}$. Tokio skaičiavimo standartinė paklaida yra apie $\frac{\hat{\sigma}}{\sqrt{2n}}$ [6].

Kyla svarbus klausimas – kiek praeities duomenų reikia imti? Jeigu nagrinėjamos daugelio dienų akcijų kainos, tai tiksliau apskaičiuojamas grąžų logaritmų kintamumas, tačiau taip įvertinamos senos, ir galbūt labai besiskiriančios nuo dabartinių, kainų kitimo tendencijos. Imant nedidelį kiekį naujausių duomenų įvertinama pati aktualiausia informacija. Tačiau ir šis atvejis nėra pats geriausias, nes pasitaiko situacijų, kai gana nedaug kintančios kokio nors aktyvo kainos viename ar keliuose intervaluose turi staigius šuolius ir kritimus. Šie nepastovumai yra tarsi išskirtys, kurios didina dispersiją ar kaip nors kitaip daro pasirinkimo sandorių įkainojimo metodų taikymą neadekvačiu. Dėl to kai kuriais atvejais išskirtys gali būti pašalintos, t.y. tam tikri intervalai neturėtų būti imami skaitinėms charakteristikoms skaičiuoti.

Tarkime, nagrinėjamas mėnesio trukmės sandoris, kurį planuojama pasirašyti vasario mėnesį (dabar). Tinkamai įvertinti grąžų logaritmų parametrus galima rinktis pastarųjų 2 mėnesių akcijų kainų duomenis. Sakykime, kad išsiaiškinta apie ženklius sezoninius akcijos nepastovumus gruodžio mėnesį. Įvertinus tai, turėtume pasirinkti nors ir trumpesnį, tačiau tą istorinių duomenų intervalą, į kurį nepatektų labiausiai nepastovus periodas gruodžio mėnesį. Nagrinėjamas sandoris truks trumpai ir tikrai nepasieks kito kainų nepastovumo periodo, taip tiksliau įvertinamas kainų kintamumas. Į šį pastebėjimą labiausiai dėmesį turėtų atkreipti analitikas, kuris siekia įkainoti pasirinkimo sandorį ar spręsti kitą uždavinį, kuris susijęs su aktyvų kainų dinamika. Nesant tokių kraštutinių situacijų bendras patarimas yra imti tokį istorinių duomenų periodą, kuriam skaičiuosime σ (dažniausiai 1 metai) arba tokį, kuris yra lygus būsimo sandorio trukmei.

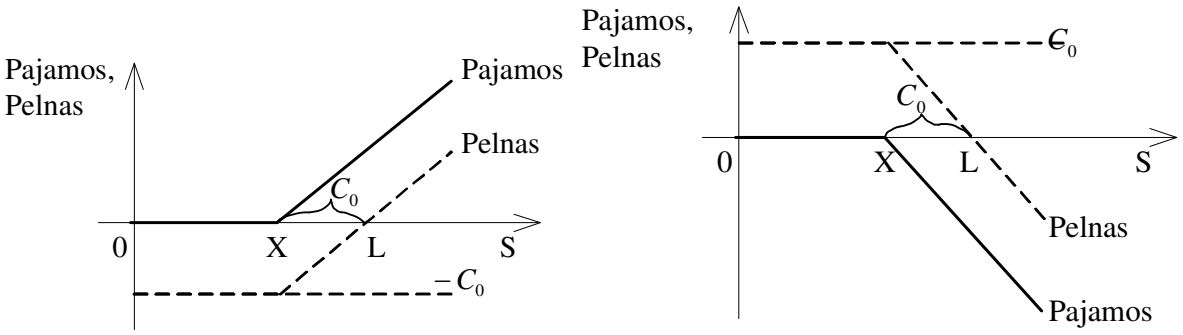
Dar viena labai svarbi problema yra laiko skaičiavimas [16]. Praktikoje priimta, kad metai turi 252 dienas, t.y. nagrinėjamos tik tos dienos, kuriomis biržoje vyksta prekyba vertybiniais popieriais, pavyzdžiui, 1 sandorio trukmės mėnuo trunka 21 dieną. Teorijoje kartais priimama, kad metai turi 360 dienų. Šiuo atveju mėnesis trunka 30 dienų. Kadangi skaičiuojame metinį aktyvų kintamumą, tai abiem atvejais sandorio trukmę, išreikštą metų dalimis gauname vienodą: $T = \frac{21}{252} = \frac{30}{360} = \frac{1}{12}$.

1.3.3. PASIRINKIMO SANDORIO VERTĖ

Pažymėkime akcijos kainą sandorio pasirašymo metu S_0, sandorio trukmę T, akcijos kainą sandorio pabaigoje S_T, ceremonijos kainą X. Pradiniu momentu S_T yra atsitiktinis dydis. Toliau bus nagrinėjamas europietiškas pasirinkimo pirkti sandoris. Šio sandorio savininkas realizuos sandorį tik tuo atveju, jeigu akcijos kaina momentu T bus didesnė už X. Priešingu atveju sandorio realizuoti neapsimoka. Sandorio pajamos momentu T:

$$C_T = \begin{cases} S_T - X, \; jei \; S_T > X; \\ 0, \; jei \; S_T \le X. \end{cases} = \max\{0; S_T - X\}. \qquad \textbf{(1.3.2)}$$

Jeigu sandorio premiją pažymėsime C_0, tai sandorio pelną bei pajamas galime pavaizduoti grafiškai:



a) b)

1.3.1 pav. Call sandorio savininko (a) ir pardavėjo (b) pajamos ir pelnas

Pasirinkimo pirkti finansinį aktyvą sandorio savininkas apriboja galimą maksimalų savo nuostolį. Pasirinkimo pirkti finansinį aktyvą sandorio pardavėjas apriboja galimą maksimalų savo pelną, tačiau nuostolis lieka neapribotas.

Pasirinkimo pirkti sandorio vertė nuliniu momentu yra diskontuota šio sandorio vertė jo pabaigoje. Kai žinomas galimų aktyvo kainų skirstinys sandorio pabaigos momentu, tai dabartinė (tikroji) pasirinkimo pirkti sandorio vertė skaičiuojama pagal 1.3.3 formulę.

$$C = \frac{1}{(1+r_f)^T} \cdot E \max\{0; S_T - X\}, \qquad \textbf{(1.3.3)}$$

čia r_f yra nerizikingoji palūkanų norma.

Amerikietiškojo tipo pasirinkimo sandoris gali būti įvykdytas bet kuriuo momentu iki jo pabaigos imtinai. Europietiškasis pasirinkimo sandoris gali būti įvykdytas tik jo pabaigoje. Akivaizdu, kad amerikietiškasis pasirinkimo sandoris yra palankesnis pirkėjui ir rizikingesnis pardavėjui, europietiškasis tokio skirtumo neturi. Tačiau taip yra tik vieno ar kelių pavienių sandorių atveju. Iš tikrųjų, amerikietiškąjį pasirinkimo pirkti sandorį nevertą įvykdyti anksčiau jo pabaigos jei tokius sandorius perkame daug kartų (nuolat). Aišku, tai galioja tik kylančios ir pakankamai stabilios rinkos sąlygomis.

1.4. PASIRINKIMO SANDORIŲ ĮKAINOJIMO METODAI

1.4.1. BLACK-SCHOLES FORMULĖ

Aštunto dešimtmečio pradžioje Fischer Black, Myron Scholes ir Robert Merton sukūrė labai reikšmingą modelį akcijų pasirinkimo sandoriams įkainoti. Už šį modelį 1997 m. R. Merton ir M.

Scholes buvo apdovanoti Nobelio ekonomikos premija. Šio modelio akcijų kainų dinamika aprašoma remiantis prielaida, kad akcijų kainų grąžų logaritmai yra pasiskirstę pagal normalųjį dėsnį [4].

Iš lognormaliojo kainų pasiskirstymo randamas ir tolygiosios (akcijos grąžos) palūkanų normos δ pasiskirstymas [6]. Įsivaizduokime, kad akcijos kaina kinta monotoniškai. Po laiko periodo T akcijos kaina nusakoma formule 1.4.1.

$$S_T = S_0 e^{\delta T}. \tag{1.4.1}$$

Iš šios lygybės išreiškus δ gaunama, kad $\delta = \frac{1}{T}\ln\frac{S_T}{S_0}$. Akivaizdu, kad δ yra pasiskirsčiusi pagal normalųjį dėsnį kaip ir grąžų logaritmai, tik su $\sqrt{T}$ mažesniu standartiniu nuokrypiu.

$$\delta \sim N\left(\mu - \frac{\sigma^2}{2}, \frac{\sigma}{\sqrt{T}}\right). \tag{1.4.2}$$

Ilgėjant sandorio laikotarpiui mažėja δ dispersija, taip yra todėl, kad investuotojas yra labiau užtikrinti gauti tam tikrą grąžą per ilgesnį laikotarpį. Tikėtina grąža per metus yra $\mu - \frac{\sigma^2}{2}$. Taip yra todėl, kad ji priklauso ne tik nuo akcijos grąžos. Kiekvienas investuotojas siekia maksimalios grąžos prie tam tikros rizikos arba atvirkščiai – mažiausios rizikos prie tam tikros grąžos (retesnis atvejis). Turėdamas galimybę investuoti ne tik į akcijas jis gali pasirinkti obligacijas, indėlį ir t.t. Tačiau dabartinė pasirinkimo sandorių įkainojimo teorija teigia, kad sandorio kaina nepriklauso nuo kainų grąžų vidurkio. Toliau prieštaros būdu bus įrodyta, kad negalima modelyje naudoti akcijų grąžų vidurkio.

Nesunku pastebėti, kad egzistuoja vidurkis μ, su kuriuo $E(S_T) = S_0 e^{\mu T}$ ir $\ln E(S_T) = \ln S_0 + \mu T$. Jeigu laipsnyje esantis μ yra apskaičiuotas iš akcijų kainų grąžų logaritmų, tai tada $E\left(\ln\frac{S_T}{S_0}\right) = \mu T$ ir $E(\ln S_T - \ln S_0) = \mu T$. Parašius vidurkių skirtumą gaunama $E(\ln S_T) = \ln S_0 + \mu T$. Tačiau $E(\ln S_T) \neq \ln E(S_T)$, nes $\ln(\cdot)$ nėra tiesinė funkcija. Gauta prieštara, kuri įrodo, kad modelyje negalima naudoti akcijos grąžos vidurkio.

Tarkime, kad akcijos kaina *S* yra aprašoma procesu:

$$dS = \mu S dt + \sigma S dz. \tag{1.4.3}$$

Išvestinio vertybinio popieriaus kaina yra pagrindinio aktyvo ir laiko funkcija. Remiantis Ito lema, išvestinio vertybinio popieriaus kaina *f* tenkina lygtį:

$$df = \left(\frac{\partial f}{\partial S}\mu S + \frac{\partial f}{\partial t} + \frac{1}{2}\frac{\partial^2 f}{\partial S^2}\sigma^2 S^2\right)dt + \frac{\partial f}{\partial S}\sigma S dz, \tag{1.4.4}$$

Lygtis (1.4.3) ir (1.4.4) taip pat užrašoma naudojant nykstamai mažus laiko pokyčius:

$$\Delta S = \mu S \Delta t + \sigma S \Delta z, \tag{1.4.5}$$

$$\Delta f = \left(\frac{\partial f}{\partial S} \mu S + \frac{\partial f}{\partial t} + \frac{1}{2} \frac{\partial^2 f}{\partial S^2} \sigma^2 S^2 \right) \Delta t + \frac{\partial f}{\partial S} \sigma S \Delta z, \tag{1.4.6}$$

čia Δf ir ΔS yra sandorio kainos ir akcijos kainos pokyčiai per nykstamai trumpą laiko tarpą Δt. Abi lygtys 1.4.3 ir 1.4.4 turi tą patį Vinerio procesą *dz*. Pasirodo, kad išvestinę finansinę priemonę, kurios kaina tenkina 1.4.4 lygtį, galima panaudoti sudarydami nerizikingą portfelį. Tam reikia parduoti vieną tokį įšvestinį vertybinį popierių (VP) ir nusipirkti $\frac{\partial f}{\partial S}$ pagrindinio aktyvo akcijų. Portfelio vertė bus: $\Pi = -f + \frac{\partial f}{\partial S} S$. Portfelio vertės pokytis per nykstamai trumpą laiko tarpą bus: $\Delta \Pi = -\Delta f + \frac{\partial f}{\partial S} \Delta S$. Jei į šią lygtį įrašysime formules 1.4.5 ir 1.4.6, tai nerizikingo portfelio vertės pokytis bus apskaičiuojamas pagal formulę 1.4.7. Lygtyje jau nėra Vinerio proceso, todėl sudarytas portfelis yra nerizikingas per laikotarpį Δt.

$$\Delta \Pi = -\left(\frac{\partial f}{\partial t} - \frac{1}{2} \frac{\partial^2 f}{\partial S^2} \sigma^2 S^2 \right) \Delta t. \tag{1.4.7}$$

Šioje vietoje reikia pabrėžti labai svarbų momentą – išvestinę vertybinę priemonę stengiamasi įkainoti taip, kad rinkoje nebūtų arbitražo galimybės. Būtent dėl to sudarytas portfelis turi uždirbti tokią pačią grąžą kaip ir trumpalaikiai nerizikingi VP su palūkanų norma *r*. Priešingu atveju egzistuotų šansas gauti garantuotą pelną su 0 – line investicija. Jei portfelio grąža didesnė negu *r*, tai galima skolintis pinigų su *r* palūkanų norma ir be rizikos iš už paskolą nusipirkto portfelio uždirbti daugiau negu *r*. Priešingu atveju galima parduoti portfelį, o už gautas pajamas pirkti nerizikingų VP. Abiem atvejais gaunamas garantuotas pelnas be rizikos. Iš čia seka, kad $\Delta \Pi = r \Pi \Delta t$. Į šią priklausomybę įrašius prieš tai išvestas Π ir $\Delta \Pi$ reikšmes, gaunama Black-Scholes diferencialinė lygtis:

$$\frac{\partial f}{\partial t} + rS \frac{\partial f}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} = rf. \tag{1.4.8}$$

Reikia pastebėti, kad šios lygties išvedimui naudotas portfelis nerizikingas buvo tik labai trumpą laikotarpį, nes *S* kinta laike, o tuo pačiu kinta ir $\frac{\partial f}{\partial S}$. Norint portfelį išlaikyti nerizikingu reikėtų nuolat keisti išvestinio ir nerizikingojo VP proporcijas portfelyje. Priklausomai nuo pagrindinio aktyvo *S*, ši lygtis gali turėti daug sprendinių. Konkretaus sprendinio radimas priklauso nuo kraštinių sąlygų. Europietiškojo pasirinkimo pirkti sandorio atveju taške $t = T$ (sandorio pabaiga) kraštinė sąlyga yra: $f = \max(S - X, 0)$, o put sandoriui: $f = \max(X - S, 0)$, kai įvykdymo kaina yra *X*. Tada atitinkamos šių sandorių vertės *C* ir *P* yra tokios:

$$C = S_0 N(d_1) - X e^{-rT} N(d_2), \tag{1.4.9}$$

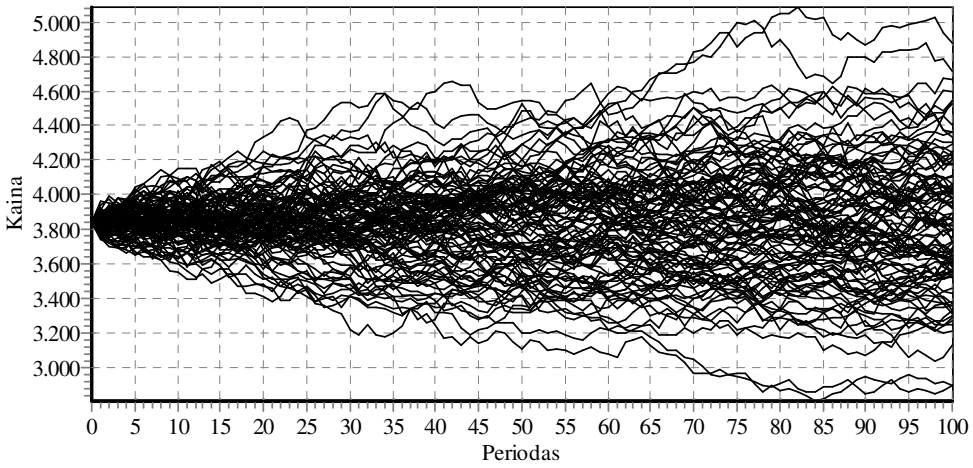
$$P = Xe^{-rT}N(-d_2) - S_0N(-d_1), \tag{1.4.10}$$

$$d_1 = \frac{\ln\frac{S_0}{X} + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}, \tag{1.4.11}$$

$$d_2 = d_1 - \sigma\sqrt{T}\ . \tag{1.4.12}$$

1.4.2. MONTE KARLO MODELIAVIMAS

Monte Karlo modeliavimas dažniausiai atliekamas tada, kai uždavinį išspręsti analitiškai yra labai sunku arba nežinoma, kaip tai padaryti. Šio metodo pritaikomumas nagrinėjamai problemai yra labai didelis. Pagrindiniai Monte Karlo modeliavimo privalumai yra tokie: nereikia sudėtingų matematinių sąryšių, realizacijas generuoti yra paprasta, siekiant tikslesnio rezultato pakanka padidinti realizacijų skaičių, dažniausiai sukurtą modelį prireikus paprasta pakeisti sudėtingesniu, jūsų modeliu bus labiau pasitikima [14]. Trūkumai yra didelės laiko sąnaudos, visada atsitiktinis rezultatas.



1.4.1 pav. Monte Karlo modeliavimo trajektorijos

Žinome, kad akcijos kainos kitimas yra Brauno judesys, aprašomas formule 1.4.5. Atsitiktinis lognormaliai pasiskirsčiusios akcijos kainos kitimas modeliojamas pagal tokią formulę 1.4.13.

$$S(t+\Delta t) = S(t)e^{\left(\delta - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t}Z}, \tag{1.4.13}$$

čia $Z \sim N(0,1)$, δ yra metinė nerizikingoji tolygioji palūkanų norma, o σ – metinis akcijos grąžos logaritmo kintamumas. Darbo metu sukurtoje programoje dydis Z generuojamas kaip 6 tolygiųjų atsitiktinių dydžių suma minus 3.

$$Z = \sum_{i=1}^{6} T_i - 3,\ T_i \sim T(0,1). \tag{1.4.14}$$

Toliau bus aptarta, kaip atliekamas modeliavimo procesas. Sandorio trukmė T dalijama į N periodų. Pradėdant nuo $t = 0$ generuojama akcijos kaina po laiko periodo Δt ir taip toliau iki sandorio pabaigos. Rezultatas bus 1 iš galimų akcijos kainos kitimo trajektorijų. Pradedant nuo pradinio

momento vėl generuojama kita trajektorija. Kuo daugiau trajektorijų bus sugeneruota, tuo tikslesnį rezultatą vėliau pavyks suskaičiuoti. 1.4.1 paveiksle pavaizduotos 100 Monte Karlo trajektorijų, gautos sukurta programa modeliuojant Canon korporacijos akcijų kainų dinamiką.

Turint sugeneruotas akcijų kainas S_T sandorio pabaigoje, randamos atitinkamos sandorio išmokos ir skaičiuojamas jų vidurkis. Apskaičiuotas dydis yra europietiškojo pasirinkimo sandorio kaina jo įvykdymo momentu. Siekdami, kad nesusidarytų arbitražo galimybė, šią kainą diskontuojame į momentą $t = 0$ su nerizikingųjų palūkanų norma δ. Formulė 1.4.15 yra europietiškojo pasirinkimo pirkti sandorio kaina.

$$C = e^{-N\cdot\Delta t\cdot\delta} E\max\left(S_T - X, 0\right). \qquad \textbf{(1.4.15)}$$

Nesunku pastebėti, kad generuojant akcijų kainas galima kaupti papildomą informaciją. Pavyzdžiui, kaupdami kainų sumas, galime kiekvienai trajektorijai sandorio pabaigoje kaip galutinę kainą S_T priskirti S_i vidurkį ($i = \overline{1, N}$). Tokią akcijos kainą sandorio įvykdymo momentu turintis pasirinkimo sandoris vadinamas Azijietišku pasirinkimo sandoriu.

1.5. SKIRSTINIŲ APROKSIMAVIMAS IR ĮVERTINIMAS

Atsitiktinio dydžio pasiskirstymą galima aproksimuoti žinomais skirstiniais sulyginant pirmuosius teorinius ir empirinius pradinius momentus. Tokiu būdu surandami nežinomi aproksimuojančio skirstinio parametrai.

1.5.1. TEIGIAMO A.D. SKIRSTINIO APROKSIMAVIMAS EKSPONENTINIŲ SKIRSTINIŲ MIŠINIU

Sakykime turime tam tikro teigiamo atsitiktinio dydžio pasiskirstymo funkciją $G(x)$. Taip pat priimkime, kad žinomi pirmieji šio skirstinio pradiniai momentai m_k, $k = 1,2,3$. Šių momentų pakanka, kad skirstinį $G(x)$ būtų galima aproksimuoti eksponentinių skirstinių mišiniu. Nagrinėkime atsitiktinį dydį

$$Y = Y_1 + Y_2, \qquad \textbf{(1.5.1)}$$

čia Y_1 ir Y_2 yra nepriklausomi a.d., pasiskirstę pagal eksponentinius dėsnius su parametrais μ_1 ir μ_2. A.d. Y tankio funkcijos išraiška surandama panaudojus sąsūkos operaciją.

$$f(x) = \int_0^{+\infty} p_{Y_1}(x-y)p_{Y_2}(y)dy = \int_0^{+\infty} \mu_1 e^{-\mu_1(x-y)}\mu_2 e^{-\mu_2 y}dy = \mu_1\mu_2\int_0^{+\infty} e^{-\mu_1(x-y)-\mu_2 y}dy =$$

$$= \mu_1\mu_2\int_0^{+\infty} e^{y(\mu_1-\mu_2)-\mu_1 x}dy = \frac{\mu_1\mu_2}{\mu_1-\mu_2}\int_0^{+\infty} e^{y(\mu_1-\mu_2)-\mu_1 x}d\left(y(\mu_1-\mu_2)-\mu_1 x\right) = \qquad \textbf{(1.5.2)}$$

$$= \frac{\mu_1\mu_2}{\mu_1-\mu_2} e^{-y(\mu_2-\mu_1)-\mu_1 x}\Big|_0^{+\infty} = \frac{\mu_1\mu_2}{\mu_1-\mu_2}\left(0 - e^{-\mu_1 x}\right) = -\frac{\mu_1\mu_2}{\mu_1-\mu_2}e^{-\mu_1 x}.$$

Suintegravus tankį gaunama pasiskirstymo funkcija $F(y)$.

$$F(y)=1-e^{-\mu_1 y}+\frac{p\mu_1}{\mu_2-\mu_1}\left(e^{-\mu_2 y}-e^{-\mu_1 y}\right). \tag{1.5.3}$$

Dabar skirstinys $G(x)$ aproksimuojamas Erlango skirstiniu taip, kad $G(x)$ ir $F(y)$ pirmieji 3 pradiniai momentai sutaptų. Reikia rasti parametrus p, μ_1 ir μ_2, su kuriais taip bus. Tam sulyginami abiejų skirstinių pradiniai momentai ir gaunama netiesinių lygčių sistema.

$$\begin{cases} \dfrac{1}{\mu_1}+\dfrac{p}{\mu_2}=m_1, \\ \dfrac{2}{\mu_1^2}+\dfrac{2p(\mu_1+\mu_2)}{\mu_1\mu_2^2}=m_2, \\ \dfrac{6}{\mu_1^3}+\dfrac{6p\left(\mu_1^2+\mu_1\mu_2+\mu_2^2\right)}{\mu_1^2\mu_2^3}=m_3. \end{cases} \tag{1.5.4}$$

Iš lygčių sistemos pirmosios ir antrosios lygybių galima rasti $F(y)$ variacijos koeficientą v_c^2.

$$v_c^2=\frac{m_2-m_1^2}{m_1^2}=\frac{\mu_2^2+p(2-p)\mu_1^2}{(\mu_2+p\mu_1)^2}. \tag{1.5.5}$$

Kadangi $(\mu_2+p\mu_1)^2\geq 4p\mu_1^2(p-1)$, nes $0\leq p\leq 1$, tai $v_c^2\geq\frac{1}{2}$. O tai parodo, kad Erlango skirstiniu galima aproksimuoti tik tokį skirstinį, kurio variacijos koeficientas yra nemažesnis už $\frac{1}{2}$.

1.5.2. TEIGIAMO A.D. SKIRSTINIO APROKSIMAVIMAS ERLANGO SKIRSTINIU

Tikimybinio skirstinio momentus generuojančią funkciją (MGF) galima skleisti Makloreno eilute. Pasirinkus pakankamą šios laipsninės eilutės narių skaičių, gaunama MGF aproksimacija. Belieka panaudojus atgręžimo formules gauti nagrinėjamo skirstinio tankio funkciją.

Funkcijos $f(t)$ Makloreno eilutė yra:

$$f(t)=f(0)+f'(0)t+\frac{f''(0)}{2!}t^2+\ldots+\frac{f^{(n)}(0)}{n!}t^n+\ldots \tag{1.5.6}$$

Erlango skirstinio MGF yra:

$$M(t)=\left(\frac{\lambda}{1-t}\right)^k,\ t<\lambda. \tag{1.5.7}$$

Nagrinėkime Erlango skirstinio MGF 2 pirmąsias išvestines.

$$\begin{cases} M'(t) = \lambda^k k; \\ M''(t) = \lambda^k k(k+1); \\ M'''(t) = \lambda^k k(k+1)(k+2). \end{cases}, \; M^{(l)}(t) = \lambda^k \frac{\prod_{n=0}^{l-1}(k+n)}{(1-t)^{k+l}}, \; M^{(l)}(0) = \lambda^k \prod_{n=0}^{l-1}(k+n). \quad \textbf{(1.5.8)}$$

Imama 3 narių MGF laipsninė eilutė.

$$M(t) \approx \lambda^k + \lambda^k kt + \frac{1}{2}\lambda^k k(k+1)t^2. \quad \textbf{(1.5.9)}$$

$$f(t) = M(it). \quad \textbf{(1.5.10)}$$

Atgręžimo formulės leidžia iš charakteristinės funkcijos gauti tankį ir atvirkščiai. Jos yra tokios:

$$f(t) = \int_{-\infty}^{+\infty} e^{itx} p(x)dx. \quad \textbf{(1.5.11)}$$

$$p(x) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-tx} f(t)dt. \quad \textbf{(1.5.12)}$$

Taikant atgręžimo formules, gaunama:

$$p(x) = \frac{1}{2\pi}\int_{-\infty}^{+\infty}\left(\lambda^k + i\lambda^k kt - \frac{1}{2}\lambda^k k(k+1)t^2\right)e^{-tx}dt =$$

$$= \frac{1}{2\pi}\lambda^k \int_{-\infty}^{+\infty}\left(1 + ikt - \frac{1}{2}k(k+1)t^2\right)e^{-tx}dt =$$

$$= \frac{1}{2\pi}\lambda^k\left(\int_0^{+\infty}(1+ikt)e^{-tx}dt + ik\int_0^{+\infty}te^{-tx}dt - \frac{1}{2}k(k+1)\int_0^{+\infty}t^2e^{-tx}dt\right) = \quad \textbf{(1.5.13)}$$

$$= \frac{1}{2\pi}\lambda^k\left(\frac{1}{x} - \frac{ik}{x^2}\int_0^{+\infty}td\left(xe^{-tx}\right) + \frac{k(k+1)}{2x^3}\int_0^{+\infty}(-tx)^2e^{-tx}d(-tx)\right) =$$

$$= \frac{1}{2\pi}\lambda^k\left(\frac{1}{x} - \frac{ik}{x^3} + \frac{k(k+1)}{2x^3}\Gamma(3)\right) = \frac{\lambda^k}{2\pi}\left(\frac{1}{x} - \frac{ik}{x^3} + \frac{k(k+1)}{4x^3}\right).$$

Tokiu būdu surandama aproksimuojanti tankio funkcija. Paklaidą būtų galima įvertinti pagal paskutinį atmestą laipsninės eilutės narį. O tam, kad charakteristinės ir tankio funkcijos būtų realios, reikia, kad aproksimuojamas a.d. būtų simetrinis.

Galima panaudoti ir kitą būdą – sulyginti kelis pirmus 2 skirstinių pradinius momentus ir rasti aproksimuojančio skirstinio parametrus, su kuriais taip bus.

$$\begin{cases} \lambda^k k = m_1; \\ \lambda^k k(k+1) = m_2; \\ \lambda^k k(k+1)(k+2) = m_3. \end{cases} \quad \begin{cases} \lambda^k k = m_1; \\ \lambda^k k^2 + \lambda^k k = m_2; \\ \lambda^k k^3 + 3\lambda^k k^2 + 2\lambda^k k = m_3. \end{cases} \quad \textbf{(1.5.14)}$$

Pažymėkime $\lambda^k = a$, $k = b$ ir $ab = c$. Netiesinė lygčių sistema supaprastėja.

$$\begin{cases} ab = m_1; \\ ab^2 + ab = m_2; \\ ab^3 + 3ab^2 + 2ab = m_3. \end{cases} \quad \begin{cases} c = m_1; \\ cb + c = m_2; \\ cb^2 + 3cb + 2c = m_3. \end{cases}$$

Gaunami tokie Erlango pasiskirstymo parametrai:

$$b(m_2 - m_1) + 3(m_2 - m_1) + 2m_1 = m_3.$$

$$k = b = \frac{m_3 - 2m_1 - 3(m_2 - m_1)}{m_2 - m_1} = \frac{m_3 + m_1 - 3m_2}{m_2 - m_1}. \qquad \textbf{(1.5.15)}$$

$$\lambda = a^{\frac{1}{k}} = \left(\frac{m_2 - m_1}{b^2}\right)^{\frac{1}{b}}.$$

1.5.3. TEIGIAMO A.D. SKIRSTINIO APROKSIMAVIMAS NORMALIUOJU SKIRSTINIU

Bet kurį a.d. iš intervalo $(-\infty; +\infty)$ galima aproksimuoti normaliuoju a.d., sulyginus šių dydžių vidurkius ir dispersijas. Tam užtenka 2 lygčių, nes normalusis pasiskirstymas turi 2 parametrus. Jeigu norima, kad sutaptų ir, pavyzdžiui, trečiasis pradinis momentas, reikia parašyti trečią lygtį kaip parodyta žemiau. Iš to seka, kad taip bus tik prie tam tikro trečiojo aproksimuojamo skirstinio momento.

$$M(t) = e^{\mu t + \frac{1}{2}\sigma^2 t^2}. \qquad \textbf{(1.5.16)}$$

$$\begin{cases} M'(t) = (\mu + \sigma^2 t) e^{\mu t + \frac{1}{2}\sigma^2 t^2} = m_1; \\ M''(t) = (\mu + \sigma^2 t)^2 e^{\mu t + \frac{1}{2}\sigma^2 t^2} + \sigma^2 e^{\mu t + \frac{1}{2}\sigma^2 t^2} = m_2; \\ M'''(t) = 2\sigma^2 (\mu + \sigma^2 t) e^{\mu t + \frac{1}{2}\sigma^2 t^2} + (\mu + \sigma^2 t)^3 e^{\mu t + \frac{1}{2}\sigma^2 t^2} + \sigma^2 (\mu + \sigma^2 t) e^{\mu t + \frac{1}{2}\sigma^2 t^2} = m_3. \end{cases} \qquad \textbf{(1.5.17)}$$

$$\begin{cases} \mu = m_1; \\ \mu^2 + \sigma^2 = m_2; \\ 2\sigma^2 \mu + \mu^3 + \sigma^2 \mu = m_3. \end{cases} \quad \begin{cases} \mu = m_1; \\ \sigma^2 = m_2 - m_1^2. \end{cases} \qquad \textbf{(1.5.18)}$$

Abu parametrai yra realieji, todėl aproksimuojamo skirstinio pradiniams momentams nėra jokių apribojimų. Aproksimacija normaliuoju skirstiniu tinka vien tik teigiamiems a.d. Todėl, pavyzdžiui paraiškų aptarnavimo laiko pasiskirstymui ši aproksimacija natinkama.

1.5.4. TEIGIAMO A.D. SKIRSTINIO APROKSIMAVIMAS GAMA SKIRSTINIU

Gama skirstinio MGF yra:

$$M(t) = (1 - \theta t)^{-k}. \qquad \textbf{(1.5.19)}$$

Skaičiuojamos MGF išvestinės ir jų reikšmės taške $t = 0$.

$$\begin{cases} M'(t) = \theta k(1-\theta t)^{-k-1}; \\ M''(t) = \theta^2 k(k+1)(1-\theta t)^{-k-2}; \\ M'''(t) = \theta^3 k(k+1)(k+2)(1-\theta t)^{-k-3}. \end{cases} \quad \begin{cases} M'(0) = \theta k = m_1; \\ M''(0) = \theta^2 k(k+1) = m_2; \\ M'''(0) = \theta^3 k(k+1)(k+2) = m_3. \end{cases} \tag{1.5.20}$$

Sulyginami pirmieji 3 teoriniai ir empiriniai pradiniai momentai:

$$\begin{cases} \theta k = m_1; \\ \theta m_1(k+1) = m_2; \\ \theta^2 m_1(k+1)(k+2) = m_3. \end{cases} \quad \begin{cases} \theta k = m_1; \\ m_1(m_1+\theta) = m_2; \\ m_1(m_1+\theta)(m_1+2\theta) = m_3. \end{cases} \tag{1.5.21}$$

$$\begin{cases} \theta = \dfrac{m_2}{m_1} - m_1; \\ k = \dfrac{m_1}{\theta} = \dfrac{m_1^2}{m_2 - m_1^2}. \end{cases} \tag{1.5.22}$$

Abu gama skirstinio parametrai turi būti teigiami. Bet modeliuojant aptarnavimo sistemą, tenka aproksimuoti teigiamą a.d., todėl:

$$\begin{cases} \dfrac{m_2}{m_1} - m_1 > 0; \\ \dfrac{m_1^2}{m_2 - m_1^2} > 0. \end{cases} \Rightarrow \begin{cases} \dfrac{m_2}{m_1} > m_1; \\ m_1^2 > m_2 - m_1^2. \end{cases} \Rightarrow \begin{cases} m_2 > m_1^2; \\ m_2 < 2m_1^2. \end{cases} \Rightarrow m_2 \in \left(m_1^2; 2m_1^2\right). \tag{1.5.23}$$

Vėlgi, gautos sąlygos aproksimuojamo skirstinio pradiniams momentams, o tai jau aproksimacijos neuniversalumas.

1.5.5. NEPARAMETRINIS TIKIMYBINIO TANKIO ĮVERTINIMAS

Nagrinėkime imtį, susidedančią iš nepriklausomų ir vienodai pasiskirsčiusių atsitiktinių dydžių x_i. Branduolinis tankio įvertis pasirinktas įvertinti dydžių x_i tikimybinį pasiskirstymą.

$$\hat{f}(x) = \frac{1}{n \cdot h} \sum_{i=1}^{n} K_h\left(\frac{x - x_i}{h}\right), \tag{1.5.24}$$

čia $K(\cdot)$ yra branduolio funkcija, *h* yra jos plotis. Kaip ir tikimybinio tankio, branduolio funkcijos integralas turi būti lygus 1, o pati funkcija turi būti neneigiama.

$$\begin{cases} \int_{-\infty}^{+\infty} K(x)dx = 1, \\ K(x) \geq 0. \end{cases} \Rightarrow \begin{cases} \int_{-\infty}^{+\infty} \hat{f}(x)dx = 1, \\ \hat{f}(x) \geq 0. \end{cases} . \tag{1.5.25}$$

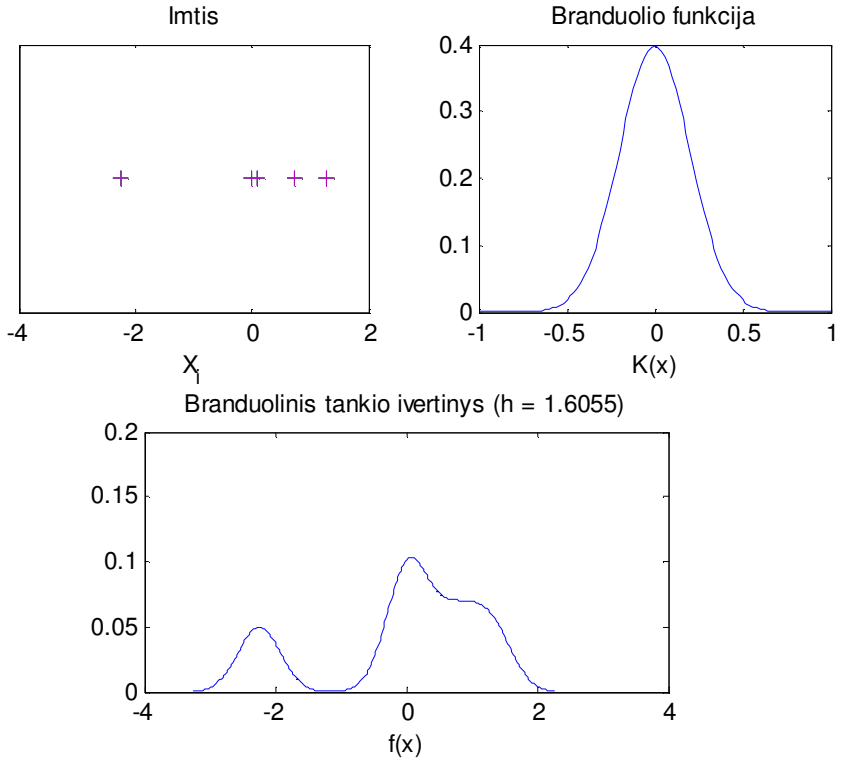
Žemiau pateiktos keletas dažniausiai naudojamų branduolio funkcijų. Pavyzdžiui, trikampė branduolio funkcija yra naudojama, kai duomenų tikimybinis tankis nėra glodus. Gauso branduolys padaro įvertį labai glodų.

$$K(x)=\begin{cases}1-|x|, & |x|\le 1,\\ 0, & |x|>1.\end{cases} \text{ (trikampis),} \qquad \textbf{(1.5.26)}$$

$$K(x)=\begin{cases}\frac{3}{4}\left(1-x^2\right), & |x|\le 1,\\ 0, & |x|>1.\end{cases} \text{ (Japaničnikovo),} \qquad \textbf{(1.5.27)}$$

$$K(x)=\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}} \text{ (Gauso).} \qquad \textbf{(1.5.28)}$$

Iš esmės toks tikimybinio tankio įvertinimas yra branduolio funkcijos priskyrimas kiekvienam x_i papildomai pridedant visų kitų priskyrimų vertes su tam tikrais svoriais. Kiekvieno kito x_j įtaka tikimybei taške x_i yra tuo mažesnė, kuo skirtumas $x_i - x_j$ yra didesnis.



1.5.1 pav. Branduolinis tikimybinio tankio įvertis

1.5.1 pav. pavaizduotas tikimybinio tankio įvertinimas imčiai iš 5 taškų naudojant Gauso branduolio funkciją. Įvertis yra visiškai glodus. Vienintelis tokio įverčio trūkumas yra būtinybė skaičiavimuose panaudoti kiekvieną imties elementą norint apskaičiuoti tikimybę tam tikrame taške.

2. TIRIAMOJI DALIS

2.1. ALTERNATYVIOS TANKIO FUNKCIJOS PARINKIMAS IR MCMC METODO KONVERGAVIMAS

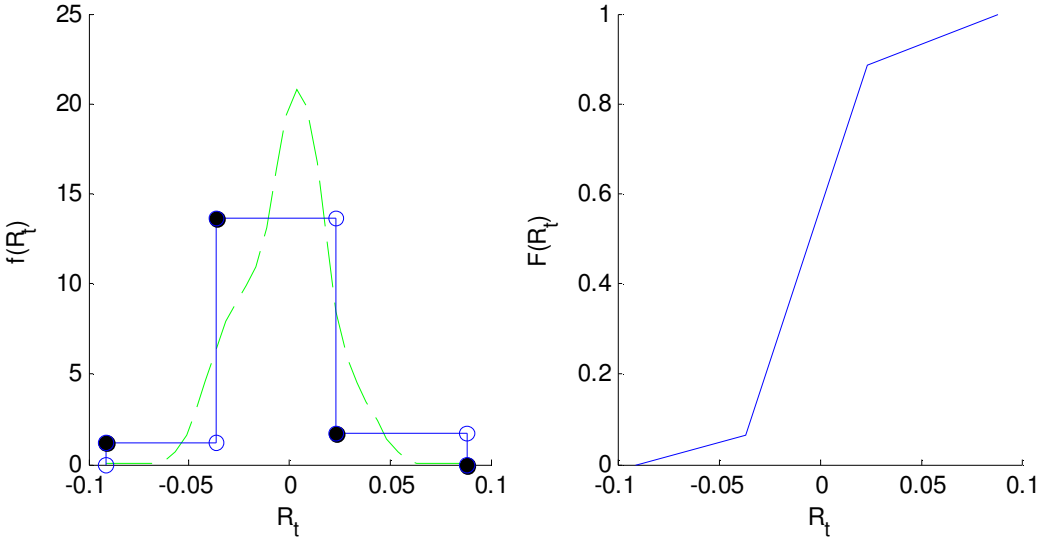
2.1.1. SPECIALI ALTERNATYVAUS PASISKIRSTYMO KONSTRAVIMO TECHNIKA

Turimas sudėtingas tikimybinis pasiskirstymas. Norint jį sumodeliuoti, reikia specialios technikos, nes nežinoma atvirkštinė pasiskirstymo funkcija arba tankio funkcijos įverčio negalima išreikšti žinomais tikimybiniais tankiais. Tokiu atveju konstruojamas tikslinio tankio įvertis ir jo modeliavimui naudojama kuri nors, tam atvejui tinkanti, technika. MCMC metodas šiuo atveju yra galima išeitis. Tačiau tiesiogiai tankio įverčiui, jei jis gautas, pavyzdžiui, skaičiuojant branduolinį tankio įvertį, MCMC taikyti nepavyks.

Pagrindinis MCMC metodo privalumas yra tas, kad juo galima sumodeliuoti norimą pasiskirstymą naudojant šį pasiskirstymą aproksimuojančią tankio funkciją, kuri turi būti panaši savo forma į tikslinį tankį. Nėra kitų reikalavimų šiai aproksimuojančiai funkcijai. Tokiu būdu alternatyvaus skirstinio sudėtingumas gali būti tiek paprastas, kiek tyrėjui to reikia.

Nagrinėkime histogramą, kuri yra santykinai greitas ir paprastas tikslinio pasiskirstymo neparametrinis įvertinimas. Todėl histogramą galima naudoti kaip alternatyvų skirstinį. Tačiau būtina įvertinti tai, kad tikslinis tankis aprašo ne diskrečius atsitiktinius dydžius, neturima imtis, kuriai galima būtų sukonstruoti histogramą.

Šiame darbe pateikta idėja, kaip naudojant dalimis tolygųjį pasiskirstymą sukonstruoti alternatyvų skirstinį branduoliniam tankio įvertiniui. Dalimis tolygusis skirstinys apibrėžtas lygtyse (2.3.1).



2.1.1 pav. Tikslinis tankio aproksimacija kaip dalimis tolygusis skirstinys

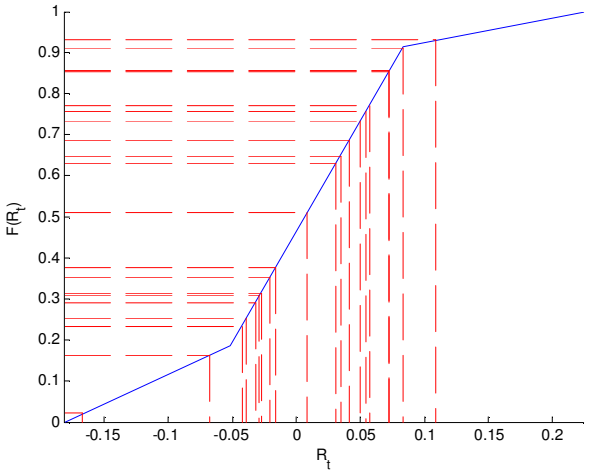
$q(x)$ pasiskirstymo funkcija gaunama įvertinant po tankio funkcija esantį plotą kiekviename iš intervalų.

$$q(x)=\begin{cases} q_1, & x_0 < x \le x_1, \\ q_2, & x_1 < x \le x_2, \\ \dots & \dots \\ q_n, & x_{n-1} < x \le x_n. \end{cases} \tag{2.1.1}$$

Tankio funkcijos ribojamas plotas turi būti lygus 1, todėl:

$$\sum_{i-1}^{n} q_i = \frac{1}{x_n - x_0}. \tag{2.1.2}$$

Toks skirstinys toliau bus laikomas alternatyviu pasiskirstymu, aproksimuojančiu tikslinį tankį. Generuoti atsitiktinius skaičius, pasiskirsčiusius pagal tokį dėsnį, yra greita ir paprasta.



2.1.2 pav. Atsitiktinių skaičių generavimas naudojant atvirkštinę skirstinio funkciją

Modeliuojant $q(x)$ reikia panaudoti paieškos procedūrą. Pirmiausiai yra sugeneruojamas $u \sim U(0;1)$. Tada reikia surasti alternatyvaus pasiskirstymo skirstinio funkcijos intervalą $(x_i, x_{i-1}]$, $i = \overline{1,n}$, kuriam priklauso u. Intervalų skaičiui esant mažam ir/arba jiems esant vienodo ilgio, šis procesas nereikalaus daug skaičiavimo laiko. Toliau u yra atvaizduojamas į x pagal $q(x)$ skirstinio funkciją kaip pavaizduota 2.1.2 paveiksle.

Naudojant $q(x)$ kaip alternatyvų skirstinį, o branduolinį tankio įvertinį kaip tikslinį tankį atsitiktinių dydžių x_i pasiskirstymas konverguos į $\hat{f}(x)$. Reikia paminėti, kad MCMC metodu generuojamų atsitiktinių skaičių x_i priėmimo į imtį tikimybė šiuo atveju yra:

$$\alpha(y|x) = \min\left(1, \frac{\hat{f}(y)q(x)}{\hat{f}(x)q(y)}\right). \tag{2.1.3}$$

Modeliavimo technika vadinama nepriklausomu Metropolis-Hastings generavimu, kai $q(x|y) = q(x)$. Nepriklausomas generatorius turi vieną svarbų pranašumą prieš tradicinį Metropolis-Hastings: seka $\{x_i\}$ praktiškai neturi atminties efekto. Kiekvienas sugeneruotas atsitiktinis dydis nepriklauso nuo ankstesnio tiesiogiai, o tik nuo pastarojo priėmimo tikimybės. Todėl yra nesvarbu,

koks buvo pasirinktas ar sugeneruotas x_0. Trumpas Metropolis-Hastings technikos aprašymas pateiktas [2].

2.1.2. MCMC METODO KONVERGAVIMO TYRIMAS

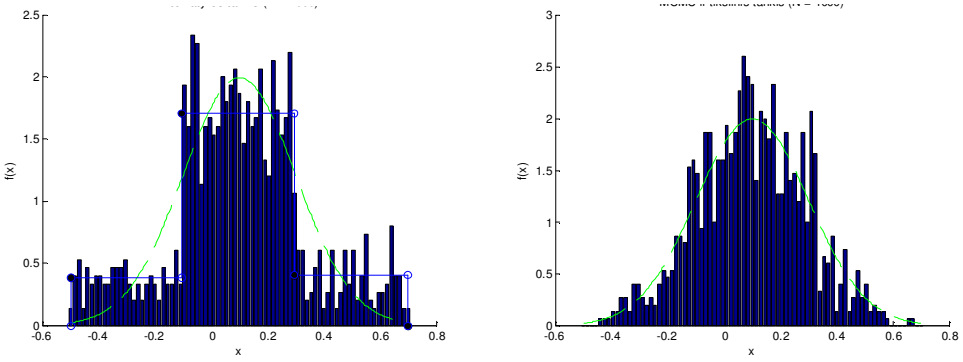
Vertinant skirtumą tarp 2 tikimybinių pasiskirstymų dažnai naudojamas šių skirstinių tankio funkcijų skirtumo integralas.

Atliekant skaitinius eksperimentus ir neparametriškai įvertinant skirstinius, patogus yra toks įvertis:

$$\Delta_h = \frac{1}{k}\sum_{j=1}^{k}\left|\pi\left(X_j\right) - h_{MCMC}\left(X_j\right)\right|. \qquad \textbf{(2.1.4)}$$

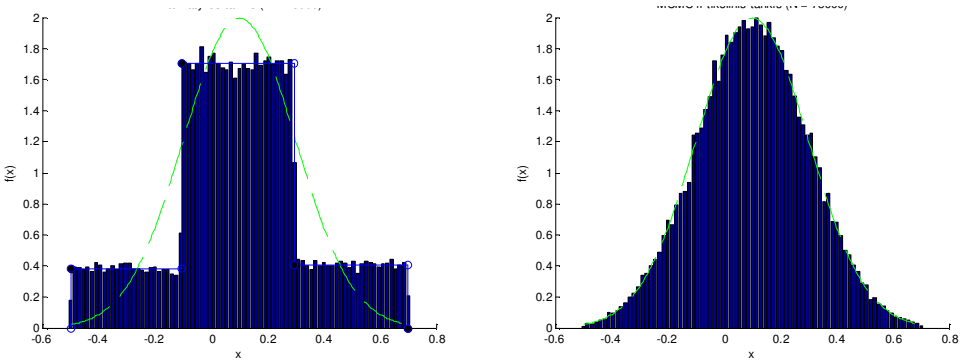
$\pi(\cdot)$ yra tikslinis tankis, $h_{MCMC}(\cdot)$ – MCMC metodu sumodeliuotos imties histograma, k – stulpelių histogramoje skaičius, o X_j nusako j-tojo stulpelio centrą.

Tyrimo metu siūloma technika modeliuotas normalusis skirstinys $N(0{,}1;0{,}2)$. Pirmiausiai tyrimas buvo atliktas su mažomis k reikšmėmis. 2.1.3 paveiksle pavaizduota MCMC metodu generuojamų atsitiktinių dydžių histograma. Histogramos stulpelių skaičius pasirinktas gerokai didesnis negu rekomenduojama teorijoje atsižvelgiant į imties dydį. Toks pasirinkimas padarytas todėl, kad būtų galima kiek įmanoma daugiau išryškinti modeliuojamo pasiskirstymo grafinį vaizdą.



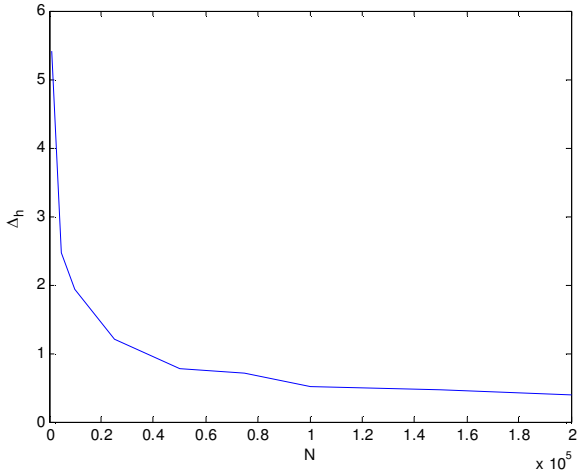
2.1.3 pav. MCMC realizavimas 1000 elementų imčiai.

Esant santykinai mažai imčiai MCMC metodas vis tiek pastebimai formuoja tikslinio tankio funkcijos formą.



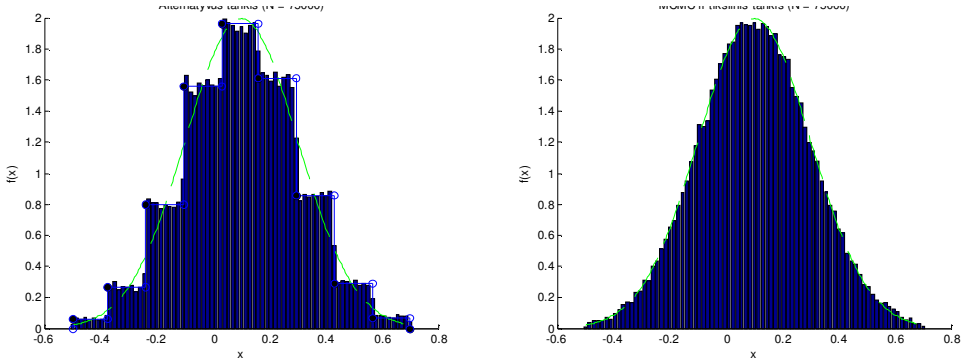
2.1.4 pav. MCMC realizavimas 75000 elementų imčiai, kai $k = 3$

Iš 2.1.4 paveikslo matyti, kad modeliuojamo dydžio apibrėžimo srities dalijimas į 3 intervalus nėra optimalus. Net prie didelio *N* metodo paklaida yra didelė.



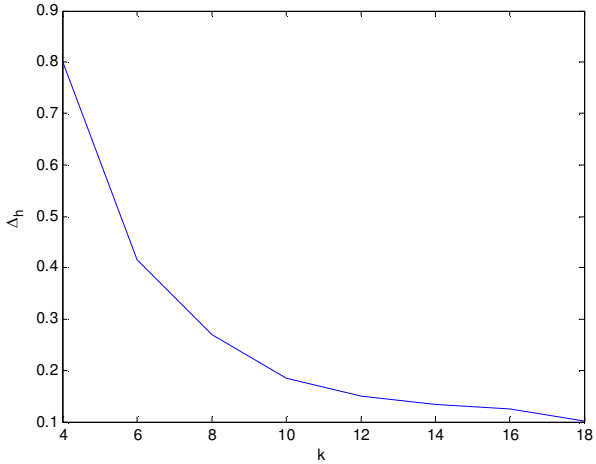
2.1.5 pav. Metodo konvergavimas *N* atžvilgiu

Įvestas konvergavimo įvertis artėja į tam tikrą reikšmę, kai *N* didėja. Nagrinėjamu atveju nuo $N = 100000$ generuojamų atsitiktinių skaičių imties didinimas jau pastebimai nedidina jų kokybės. Ją padidinti belieka tik didinant *k*.



2.1.6 pav. MCMC realizavimas 75000 elementų imčiai, kai $k = 9$

Atliekant MCMC metodo konvergavimo *k* atžvilgiu tyrimą iš buvo modeliuojama 75000 atsitiktinių dydžių.



2.1.7 pav. Metodo konvergavimas *k* atžvilgiu

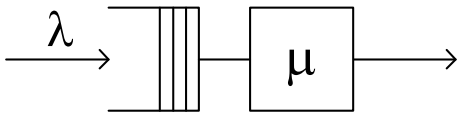
Didinant alternatyvaus pasiskirstymo tankio funkcijos intervalų skaičių mažėja skirtumai tarp šių intervalų kraštuose apskaičiuojamų alternatyvaus ir tikslinio tankių reikšmių. Atlikus tyrimą įsitikinta, kad *k* didinimas spartina konvergavimą į tikslinį tankį.

Įvertinant atliktų tyrimų rezultatus darytina išvada, kad geriausios kokybės atsitiktiniai skaičiai gaunami generuojant ne mažiau 100000 elementų imtį, o alternatyvų skirstinį formuojant apibrėžimo srityje, padalintoje į keliolika intervalų. Esant mažos imties poreikiui būtina rinktis kuo didesnį apibrėžimo srities padalijimo intervalų skaičių.

2.2. APTARNAVIMO SISTEMOS MODELIS

2.2.1. TIRIAMO MODELIO APRAŠYMAS

Darbo metu tirta aptarnavimo sistema be apribojimo eilės dydžiui, pavaizduota 2.2.1 paveiksle. Sakykime, kad ateinančių paraiškų srautas yra Puasoninis, t.y. laikas tarp gretimų paraiškų atėjimo į sistemą momentų yra pasiskirstęs pagal eksponentinį dėsnį. Aptarnavimo srautas taip pat Puasoninis. Sistema bus sumodeliuota tarus, kad paraiškų aptarnavimo srauto pasiskirstymas yra sudėtingas ir nežinoma kaip jį modeliuoti tiesiogiai.



2.2.1 pav. Tiriama aptarnavimo sistema

Praktikoje paraiškų aptarnavimo ir / arba atėjimo srautai gali turėti sudėtingas skirstinio arba tankio funkcijas. Aptarnavimo srautui generuoti taikysime Markovo grandinių Monte Karlo metodą. Tokiu būdu bus gauta modeliavimo priemonė, skirta atvejui, kai paraiškų aptarnavimo srautas turi sudėtingą pasiskirstymą. Kadangi yra žinomos formulės, pagal kurias randame paprasčiausios eilių sistemos charakteristikas, tai galėsime jas palyginti su sumodeliuotomis. Šis palyginimas dažnai vadinamas modelio kalibravimu. Žinodami, kad modelis veikia gerai, galėsime jį taikyti atvejui, kai neturime sistemos charakteristikų apskaičiavimo formulių ir gausime jas eksperimentiškai.

Tarkime, kad tiriame sistemą, kai laikas tarp ateinančių paraiškų turi pasiskirstymo funkciją

$$F_p(x) = 1 - e^{-0,5x},\ x > 0.$$

Paraiškų aptarnavimo trukmės tegul būna pasiskirstę pagal (priimta prielaida, kad šis pasiskirstymas yra sudėtingas):

$$F_a(x) = 1 - e^{-0,7x},\ x > 0.$$

Tokios eilių sistemos charakteristikos pateiktos 2.2.1 lentelėje.

2.2.1 lentelė

Eilių sistemos charakteristikos

Sistemos charakteristika	Reikšmė
Vidutinis laikas tarp 2 ateinančių paraiškų	2
Vidutinis paraiškos aptarnavimo laikas	1,4286

Vidutinis paraiškų skaičius sistemoje	2,5
Vidutinis paraiškų skaičius eilėje	1,7857
Vidutinis paraiškos buvimo eilėje laikas	3,5714
Vidutinis paraiškos buvimo sistemoje laikas	5
Tikimybė, kad kanalas užimtas	0,7143

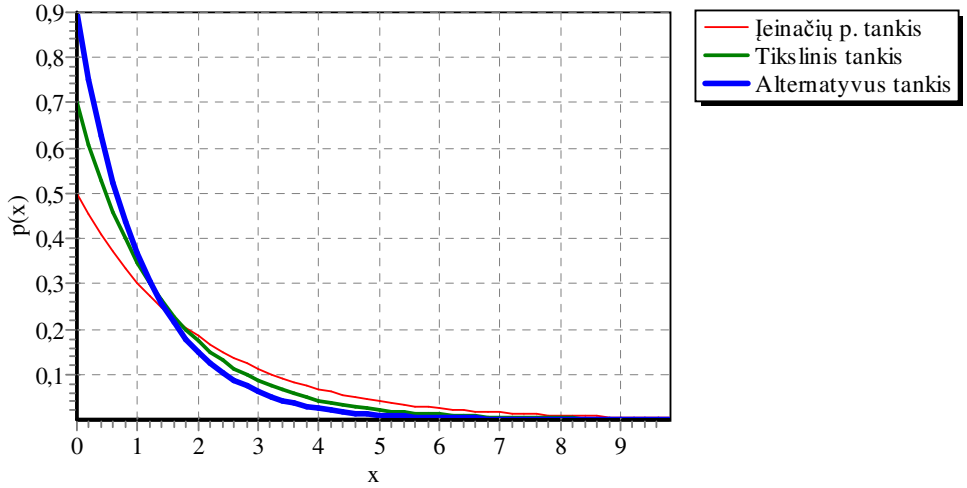
Kitame skyriuje šios charakteristikos bus surastos taikant MCMC metodą. Tam reikia „sudėtingą“ $F_a(x)$ modeliuoti naudojant į ją forma panašią skirstinio funkciją $F_q(x)$.

2.2.2. *M/G/1* SISTEMOS SU EILE MODELIAVIMO REZULTATAI

Įvertinama aptarnavimo srauto skirstinio tankio funkcijos forma ir pasirenkama:

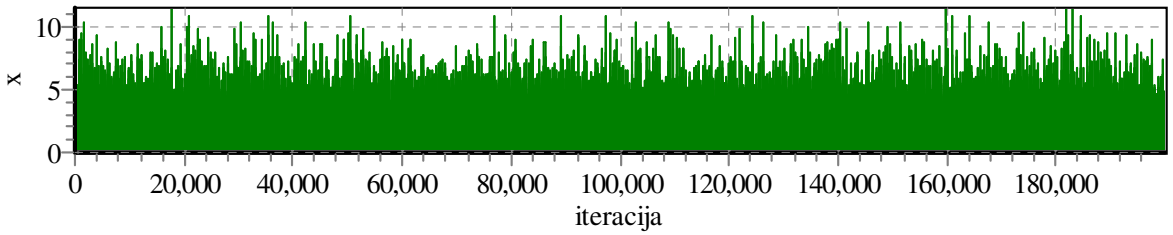
$$F_q(x) = 1 - e^{-0,9x},\ x > 0.$$

Tankio funkcijų palyginimas pateiktas 2.2.2 paveiksle.



2.2.2 pav. Tankio funkcijų palyginimas

Modeliuoti galima 2 variantus: laiką arba paraiškų skaičių. Galima pasirinkti bet kurį iš jų, nes bet kokiu atveju bet kuriam iš jų didėjant, sistemos charakteristikos artėja prie vidutinių. Sumodeliuokime eilių sistemos procesą, kai į sistemą ateina 200000 paraiškų ir kiekviena iš jų yra aptarnaujama. Taikant MCMC nepriklausomojo generavimo metodą, gaunamas aptarnavimo trukmių atsitiktinis procesas, pavaizduotas 2.2.3 paveiksle.



2.2.3 pav. Aptarnavimo srautas

Minėtas atsitiktinis procesas yra Markovo grandinė, todėl yra iteracijų, kuriose proceso reikšmė nepakinta (grandinėje neįvyksta šuolis). Be to šį procesą sudarantys atsitiktiniai dydžiai yra priklausomi. Galima būtų manyti, kad būtent priklausomumas, kaip nekokybiško atsitiktinių skaičių

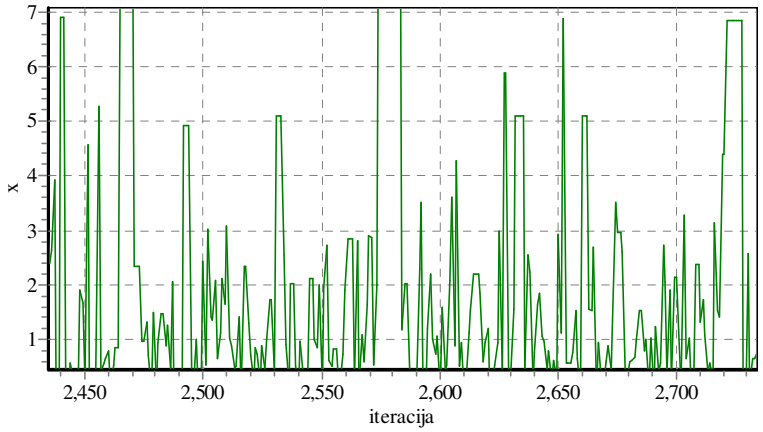
generavimo analogas galėtų lemti klaidingus sistemos įverčių skaičiavimo iš modeliavimo rezultatų radimus. Iš tikrųjų, sistemos charakteristikų įverčiai šiuo atveju gaunami visiškai netikslūs. Jie pateikti 2.2.2 lentelėje.

2.2.2 lentelė

Eilių sistemos charakteristikos

Sistemos charakteristika	Reikšmė	
	Tiksli	Sumodeliuota
Vidutinis laikas tarp 2 ateinančių paraiškų	2	1,9956
Vidutinis paraiškos aptarnavimo laikas	1,4286	1,4364
Vidutinis paraiškų skaičius sistemoje	2,5	4,5978
Vidutinis paraiškų skaičius eilėje	1,7857	3,8780
Vidutinis paraiškos buvimo eilėje laikas	3,5714	7,7392
Vidutinis paraiškos buvimo sistemoje laikas	5	9,1756
Tikimybė, kad kanalas užimtas	0,7143	0,7198

Idėja apie priklausomos imties pateikimą paraiškų aptarnavimo skirstiniui yra teisinga. Bet tai dar ne viskas, tikroji nesėkmingo modeliavimo priežastis yra giliau. Kadangi aptarnavimo srautas yra Markovo grandinė, tai, kaip minėta, pasitaiko greta esančių vienodų šio proceso reikšmių, o jei jos dar yra didelės, tai iš karto tam tikru laiko momentu sistemoje sparčiai pradeda kauptis eilė, kuri po to nyksta itin lėtai (2.2.5 pav.). Taip yra dėl to, kad eksponentinis skirstinys yra apibrėžtas teigiamoje realiųjų skaičių pusašėje ir, nors su maža tikimybe, apima ir dideles reikšmes.

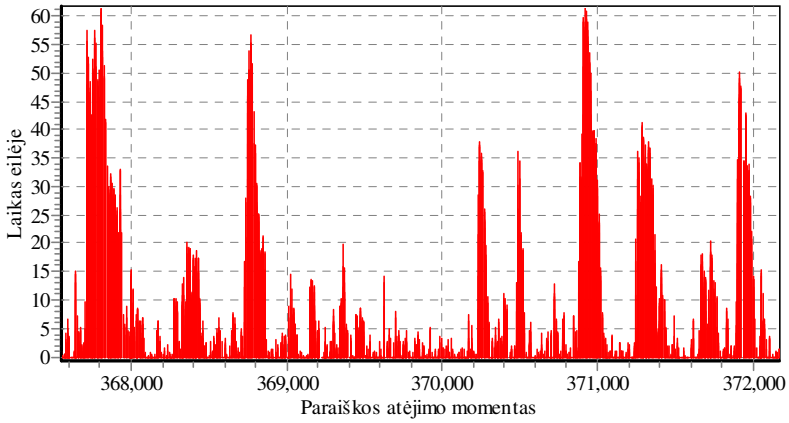


2.2.4 pav. Šuoliai MCMC metodu gautoje Markovo grandinėje

Kita vertus, turi būti labai maža tikimybė, kad eksponentinis skirstinys sugeneruos 2 dideles reikšmes iš eilės. Iš čia galima konstatuoti, kad tarsi nagrinėjame „kraštinę“ proceso realizaciją ir gauname eilių charakteristikas daug didesnes, negu vidutines. Šios išvados, aišku, tinka bet kokiam skirstiniui, kurio didesnioji tikimybinė masė koncentruota pasiskirstymo intervalo pradžioje.

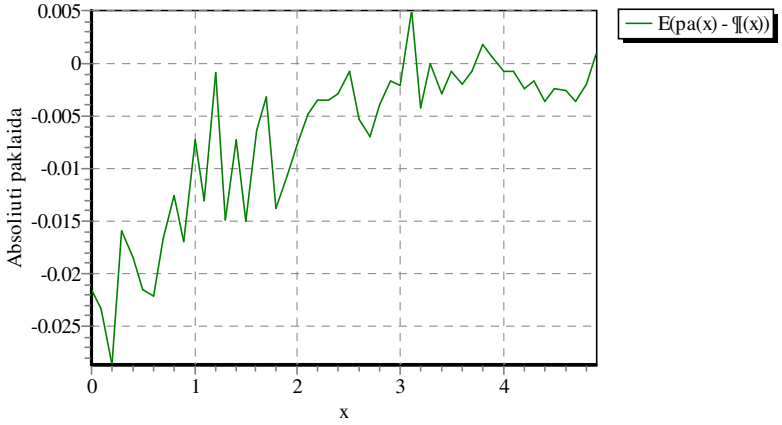
Turimos 2 priežastis: priklausoma imtis ir Markoviška proceso prigimtis, kurios reikalauja metodo tobulinimo. O jis atliekamas labai paprastai. Reikia gautą imtį tiesiog sumaišyti atsitiktinai, skirstinys nepakis, o proceso dvi gretimos reikšmės bus nepriklausomos ir nebus Markovo grandinės (konstatuoto eilių kaupimosi ir lėto mažėjimo). Dėl konkrečios programinės eilių sistemos modeliavimo realizacijos buvo pasirinkta imti kas 5 – ą atsitiktinį dydį iš MCMC nepriklausomojo

generatoriaus gautos Markovo grandinės. Akivaizdu, kad tokiu būdu Markovo grandinė yra išardoma, vien dėl to, kad nelieka priklausomumo (ar bent jau jis tampa daug silpnesnis).



2.2.5 pav. Staigūs eilių padidėjimai aptarnavimo sistemoje

Šias pastabas svarbu pastebėti, nes aptarto modeliavimo metu gautas gan tikslus paraiškų aptarnavimo skirstinys, kuris nieko apie reikiamus metodo tobulinimus neakcentuoja. Jeigu intervalą $[0;5]$ padalinsime į 50 dalių ir gautuose taškuose paskaičiuosime skirtumus tarp aptarnavimo srauto tankio ir MCMC metodu gautos Markovo grandinės histogramos bei rasime jų vidurkį, tai gausime, kad vidutiniškai bet kuriame intervalo $[0;5]$ taške sumodeliuotas tankis skiriasi nuo tikslinio tankio per $-0{,}007111$. Žemiau pateiktas absoliučių paklaidų grafikas.



2.2.6 pav. Absoliuti paklaida tarp tikslinio tankio ir Markovo grandinės tankio

Iš 2.2.6 paveikslo matyti, kad absoliuti paklaida didžiausia ten, kur alternatyvus tankis daugiau skiriasi nuo tikslinio tankio. Todėl svarbu alternatyvų skirstinį parinkti kuo panašesnį į tikslinį, kad gautume kuo didesnį aproksimavimo tikslumą. Remiantis atliktais tyrimais galima daryti rekomendaciją pasirinkti šiek tiek sunkesnę uodegą negu tikslinis turintį alternatyvų tankį, negu tą tikslinio tankio aproksimaciją, kurios uodega šiek tiek lengvesnė. Tada minėtas paklaidos įvertinimas gaunamas mažesnis.

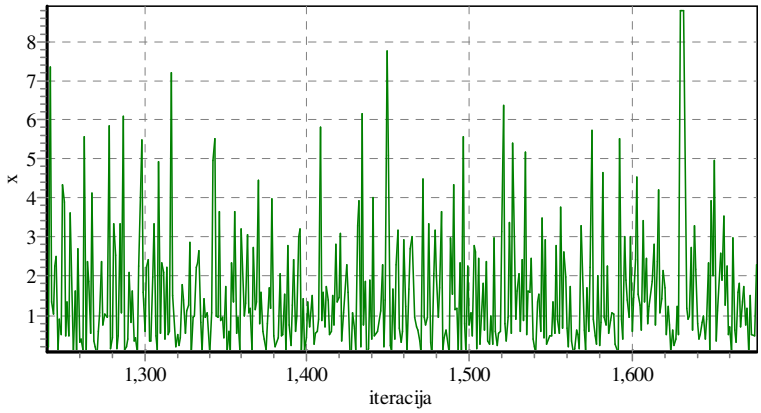
Pritaikykime anksčiau minėtą algoritmo patobulinimą ir atlikime modeliavimą iš naujo. Imkime vėl 200000 paraiškų atvejį. Gaunamos daug tikslesnes sistemos charakteristikos.

2.2.3 lentelė

Eilių sistemos charakteristikos

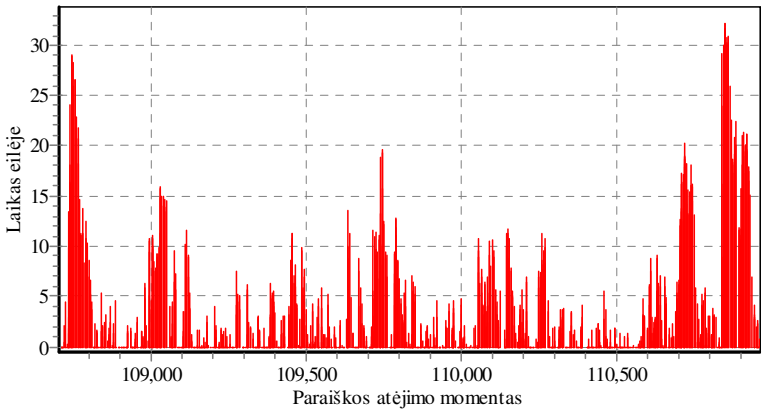
Sistemos charakteristika	Reikšmė	
	Tiksli	Sumodeliuota
Vidutinis laikas tarp 2 ateinančių paraiškų	2	1,9982
Vidutinis paraiškos aptarnavimo laikas	1,4286	1,4283
Vidutinis paraiškų skaičius sistemoje	2,5	2,7118
Vidutinis paraiškų skaičius eilėje	1,7857	1,9971
Vidutinis paraiškos buvimo eilėje laikas	3,5714	3,9906
Vidutinis paraiškos buvimo sistemoje laikas	5	5,4188
Tikimybė, kad kanalas užimtas	0,7143	0,7148

Taip pat šiuo atveju nėra tokio staigaus eilių augimo, kuris, remiantis pateiktais samprotavimais, yra nepagrįstas (2.2.7 pav.).



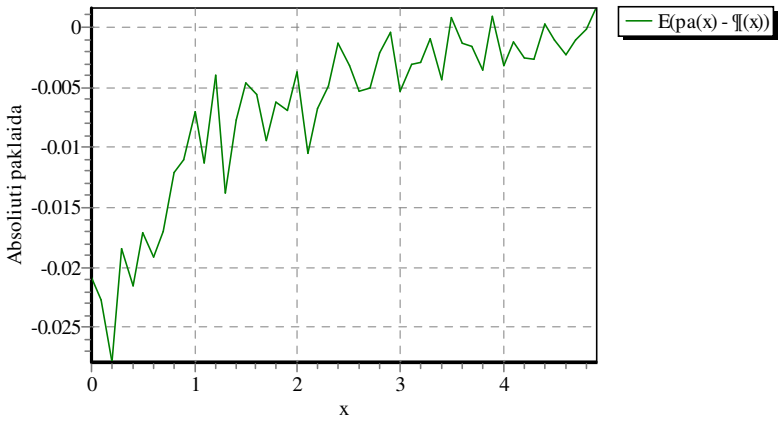
2.2.7 pav. Kokybiškesnė atsitiktinių skaičių seka

Iš Markovo grandinės imant kas penktą būseną gautame atsitiktiniame procese dažniau vyksta šuoliai, gaunama daugiau skirtingų atsitiktinių dydžių reikšmių. Tokį atsitiktinį procesą sudarantys a.d. yra vadinami kokybiškesniais.



2.2.8 pav. Adekvatus laiko eilėje kitimas

Metodo patobulinimas taip pat leido gauti ne tik tikslesnius sistemos charakteristikų įverčius, bet ir sugeneravo kiek tikslesnį skirstinį. Dabar vidutinis skirtumas taške x tarp tikslinio ir Markovo grandinės tankio yra $-0{,}006838$. Šis paklaidų skirtumas yra sumažinto Markovo grandinės proceso, kaip atsitiktinio dydžio, diskretiškumo pasekmė.



2.2.9 pav. Absoliuti paklaida tarp tikslinio ir Markovo grandinės tikimybinių tankių

2.2.3. ALTERNATYVAUS SKIRSTINIO PARINKIMAS NAUDOJANT APROKSIMACIJĄ ERLANGO SKIRSTINIU

Nagrinėkime tą pačią aptarnavimo sistemą, kaip ir 2.2. skyrelyje. Bus naudojama eksponentinio skirstinio pradinių momentų radimo formulė:

$$m_k = E\left(X^k\right) = \frac{k!}{\lambda^k},\ k > 0.$$

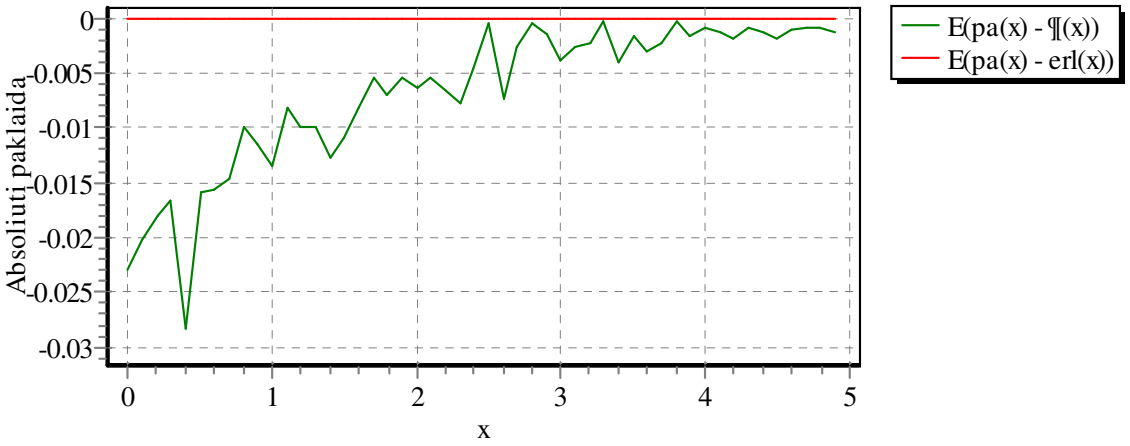
Sulyginsime 3 pirmuosius paraiškų aptarnavimo srauto ir Erlango skirstinio pradinius momentus. Gausime aproksimuojantį Erlango skirstinį ir įvykdysime modeliavimą naudodami MCMC metodą. Tokiu būdu bus surastos empirinės aptarnavimo sistemos charakteristikos. Vėl modeliuojama 200000 paraiškų, bet netaikomas MCMC nepriklausomo generatoriaus patobulinimas. Modeliavimo rezultatai pateikti 2.2.4 lentelėje.

2.2.4 lentelė

Eilių sistemos charakteristikos

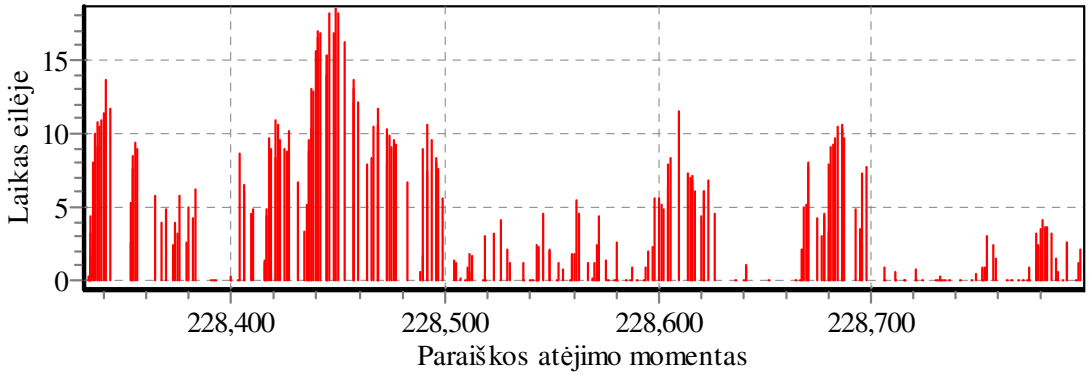
Sistemos charakteristika	Reikšmė	
	Tiksli	Sumodeliuota
Vidutinis laikas tarp 2 ateinančių paraiškų	2	1,9956
Vidutinis paraiškos aptarnavimo laikas	1,4286	1,4267
Vidutinis paraiškų skaičius sistemoje	2,5	2,4909
Vidutinis paraiškų skaičius eilėje	1,7857	1,7760
Vidutinis paraiškos buvimo eilėje laikas	3,5714	3,5443
Vidutinis paraiškos buvimo sistemoje laikas	5	4,9710
Tikimybė, kad kanalas užimtas	0,7143	0,7149

Gavome gan tikslius įverčius, nors netaikėme Markovo grandinės pakeitimų. Taip atsitiko todėl, kad Erlango aproksimacija buvo itin tiksli ir alternatyvus skirstinys beveik sutapo su tiksliniu skirstiniu. Nepaisant to, MCMC metodas labai padidino aproksimavimo paklaidas. Tai matome iš 2.2.10 paveikslo. Vidutinis MCMC sugeneruotos Markovo grandinės ir tikslinio tankio funkcijos skirtumas yra $-0{,}006811$, o tarp Erlango aproksimacijos ir tikslinio tankio tik $-4{,}90894 \cdot 10^{-6}$.



2.2.10 pav. Tikslinio tankio aproksimavimo ir MCMC modeliavimo absoliučios paklaidos

Dėl tokio tikslaus aproksimavimo Markovo grandinėje praktiškai nuolat vyko perėjimai. Kitaip tariant, nepriklausomas MCMC modeliavimas nėra geriausias pasirinkimas šioje situacijoje. Nagrinėjamą aptarnavimo sistemą vertėtų modeliuoti tiesiogiai, naudojant Erlango funkcijos atvirkštinę pasiskirstymo funkciją.



2.2.11 pav. Paraiškų laikas eilėje

Atlikime tiesioginį eilių sistemos modeliavimą. Gauname tokias sistemos charakteristikas:

2.2.5 lentelė

Eilių sistemos charakteristikos

Sistemos charakteristika	Reikšmė	
	Tiksli	Sumodeliuota
Vidutinis laikas tarp 2 ateinančių paraiškų	2	1,9939
Vidutinis paraiškos aptarnavimo laikas	1,4286	1,4326
Vidutinis paraiškų skaičius sistemoje	2,5	2,5837
Vidutinis paraiškų skaičius eilėje	1,7857	1,8652
Vidutinis paraiškos buvimo eilėje laikas	3,5714	3,7191
Vidutinis paraiškos buvimo sistemoje laikas	5	5,1518
Tikimybė, kad kanalas užimtas	0,7143	0,7185

Rezultatai yra labiau nepastovūs, jei modeliuojama tik 1 kartą, tačiau nedaug skiriasi nuo tikslių reikšmių. Laimėta daug skaičiavimų laiko, nes nereikėjo skaičiuoti priėmimo tikimybės ir pan., o užteko tik Erlango skirstinio kvantilio.

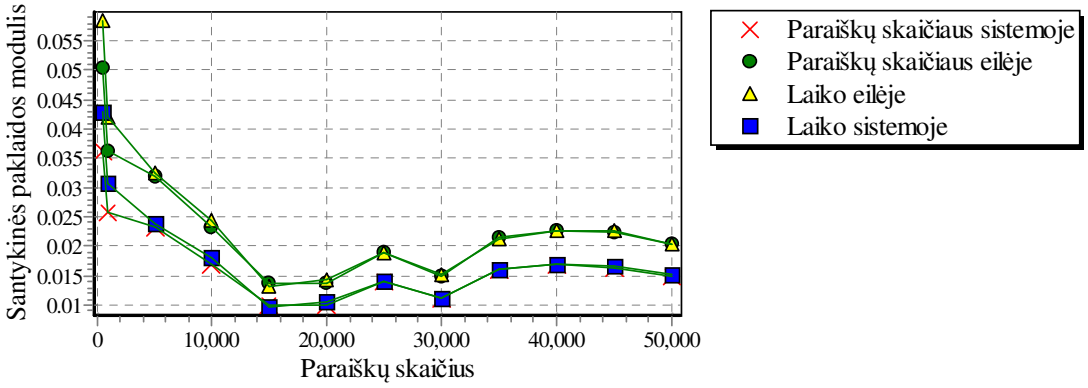
2.2.4. *M/G/1* SISTEMOS SU EILE MODELIAVIMO ADEKVATUMAS

Nagrinėjamos sistemos (2.2.1 pav.) charakteristikų skaičiavimo tikslumu laikysime jų santykinę paklaidą. Monte Karlo metodu atlikus stochastinės sistemos modeliavimą kelis kartus, gaunami empirinių charakteristikų vidurkiai μ bei standartiniai nuokrypiai σ nuo teorinių reikšmių. Kadangi standartinis nuokrypis atitinka įverčio absoliučią paklaidą, tai santykine paklaida galima laikyti dydį $\varepsilon = \sigma/\mu$.

Tyrimui pasirinkta eilių sistema, į kurią paraiškos ateina su intensyvumu $\lambda = 0{,}4$, o jų aptarnavimo intensyvumas turi lognormalųjį skirstinį su parametrais $\mu = 0{,}4$, $\sigma = 0{,}7$. Atitinkamos skirstinių funkcijos:

$$F_p(x) = 1 - e^{-\lambda \cdot x},\ F_a(x) = \Phi\left(\frac{\ln(x) - \mu}{\sigma}\right). \qquad \textbf{(2.4.1)}$$

2.2.12 paveiksle matome aprašytos eilių sistemos charakteristikų apskaičiavimo santykines paklaidas, kai sumodeliuojama nuo 500 iki 50000 ateinančių paraiškų. Šie rezultatai parodo vidutinę sistemos modeliavimo trukmę, su kuria gaunami tam tikro tikslumo charakteristikų įverčiai. Atlikus modeliavimą pastebėta ir įsitikinta, kad sistemos charakteristikų tikslumas yra tarpusavyje priklausomas.

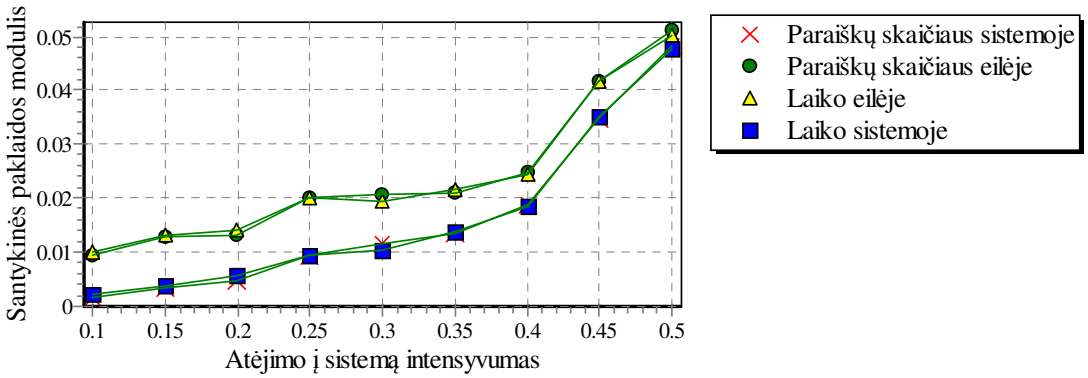


2.2.12 pav. ε modulių priklausomybė nuo atėjusių paraiškų skaičiaus

Modeliuojant įvairias eilių sistemas nustatyta, kad jų charakteristikos randamos tuo tiksliau, kuo santykinai labiau skiriasi paraiškų atėjimo ir aptarnavimo srautai. Iš tikrųjų, paraiškų pagausėjimas neturi daug įtakos, jei jos sparčiai aptarnaujamos, o priešingu atveju – pradeda kauptis eilės.

2.2.13 paveiksle pateiktos nagrinėjamos sistemos charakteristikų santykinės paklaidos, kai jų atėjimo į sistemą intensyvumas $\lambda \in [0{,}1;5]$.

Vėl pastebima paklaidų priklausomybė. Taip pat matoma didelė santykinė paklaida, kai paraiškų aptarnavimo ir atėjimo į sistemą intensyvumai (aptarnavimo intensyvumas nagrinėjamu atveju $e^{-(\mu+0{,}5\sigma^2)} \approx 0{,}52$) yra panašūs. Taip atsitinka dėl to, kad sąlyginai trumpo modeliavimo laiko nepakanka nusistovėti eilėms.



2.2.13 pav. ε modulių priklausomybė nuo λ (sumodeliuota 20000 paraiškų)

Modeliuojant įvairias eilių sistemas nustatyta, kad jų charakteristikos randamos tuo tiksliau, kuo santykinai labiau skiriasi paraiškų atėjimo ir aptarnavimo srautai. Iš tikrųjų, paraiškų pagausėjimas neturi daug įtakos, jei jos sparčiai aptarnaujamos, o priešingu atveju – pradeda kauptis eilės. Tokioje situacijoje aptarnavimo sistema dirba beveik pilnu pajėgumu.

2.3. AKCIJŲ KAINŲ DINAMIKOS MODELIAVIMAS

2.3.1. ALTERNATYVIOS TANKIO FUNKCIJOS MCMC METODUI PARINKIMAS

Klasikinis akcijų kainų modelis turi keletą prielaidų apie nagrinėjamus finansinius duomenis. Todėl jo negalima taikyti norint modeliuoti akcijos kainą, kai akcijų grąžos nėra pasiskirsčiusios pagal lognormalųjį skirstinį.

Yra keletas metodų, skirtų modeliuoti dydžius, kurie pasiskirstę pagal sudėtingus skirstinius, tačiau šie metodai yra specializuoti skirtingose srityse. Šio darbo tikslas yra pateikti universalią akcijų kainų modeliavimo techniką. Ši technika sudaryta iš specialių skaitinių metodų ir tinka bet kokiems empiriniams duomenims. Bendros metodikos, kaip parinkti alternatyvų tankį tiksliniam tankiui nėra, tik atskirtos rekomendacijos [11]. Darbe panaudotas dalimis tolygiojo skirstinio konstravimas pasirodė tinkamas šiaip problemai spręsti.

Markovo grandinių Monte Karlo metodas buvo panaudotas modeliuojant empirinį akcijos kainos skirstinį. Technika yra universali ir jos įgyvendinimui pakanka žinoti modeliuojamo pasiskirstymo tankio funkcijos reikšmę bet kuriame taške. Dar daugiau, MCMC buvo sėkmingai pritaikytas vieno faktoriaus palūkanų normos modeliams [3]. Tai taip pat pagrindžia MCMC pasirinkimą universalios akcijų kainų modeliavimo technikos kūrimui.

Yra svarbu kuo tiksliau įvertinti empirinį akcijos grąžos pasiskirstymą. Tai atlikta surandant branduolinį tankio įvertį. Buvo sugalvotas šių dviejų metodų apjungimas, kuris leido kuriamai metodikai būti taikomai bet kokiems finansiniams duomenims.

2.3.2. AKCIJOS KAINŲ GRĄŽŲ TANKIO FUNKCIJOS ĮVERTINIMAS

Šiame skyriuje pateikta metodika, skirta modeliuoti akcijų kainas arba kitus atsitiktinius dydžius (metodas yra universalus) nežinant jų analitinių skirstinio (arba tankio) funkcijų.

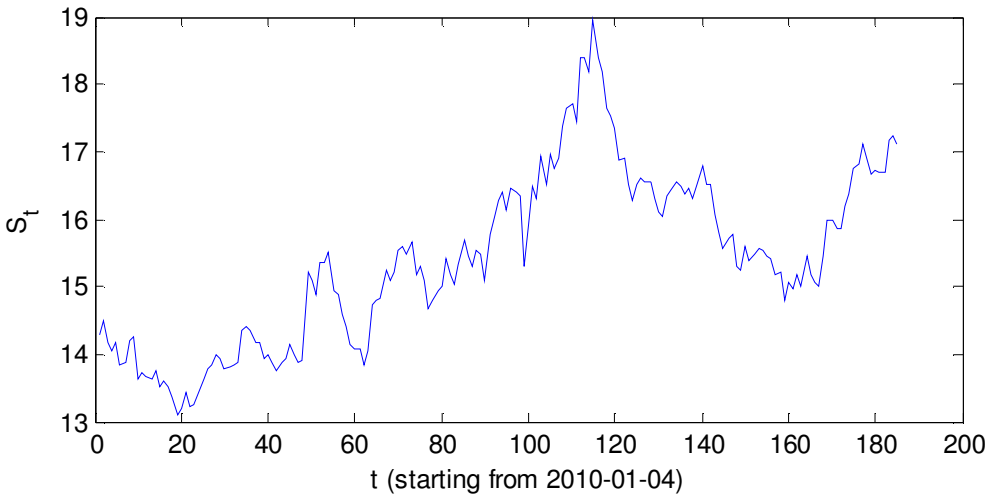
Pirmiausiai turi būti atliktas branduolinis tankio įvertinimas ir tokiu būdu sukonstruotas akcijos kainos grąžos tikimybinio tankio įvertis. Šioje vietoje galima diskusija dėl tokio įverčio tikslumo, tačiau šiame darbe jis priimamas kaip tikslus.

Taip pat nėra būtinybės galvoti apie analitines funkcijas, kurios geriausiai tinka kuriuo nors konkrečiu atveju. Apskritai nereikia galvoti apie tankio funkcijų formą, ji suformuojama apskaičiuojant tankio įvertį. Vienintelis diskusijų objektas lieka branduolio funkcijos plotis, kurio optimaliai reikšmei surasti teorijoje yra siūlomos kelios formulės [13].

2.3.3. MODELIO ADEKVATUMO TYRIMAS

Kiekvienas modelis turi pateikti adekvačius rezultatus ir būti sulyginamas su kitais žinomais modeliais ar technikomis. Kaip minėta, modelio taisymas ir tobulinimas siekiant jį tokiu padaryti vadinamas jo kalibravimu. Šiuo atveju, nauja akcijų kainų modeliavimo technika turi duoti tokius pačius arba panašius rezultatus, kaip ir Monte Karlo modeliavimas, kai akcijų kainų grąžos yra pasiskirstę pagal lognormalųjį skirstinį.

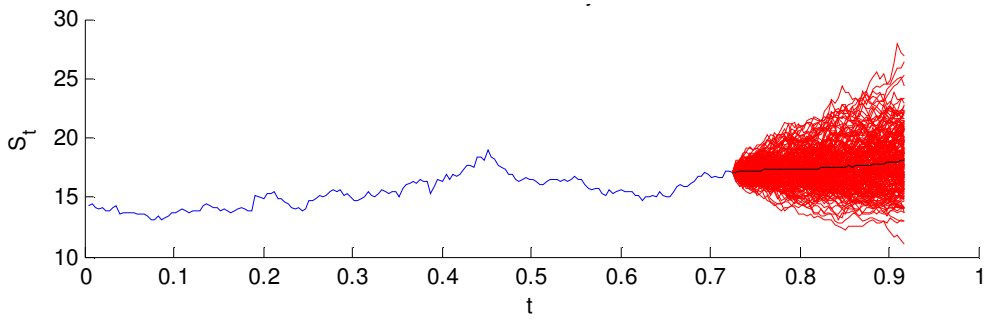
Prieš atliekant modeliavimą, būtina patikrinti hipotezę apie akcijų kainų grąžų logaritmų normalumą.



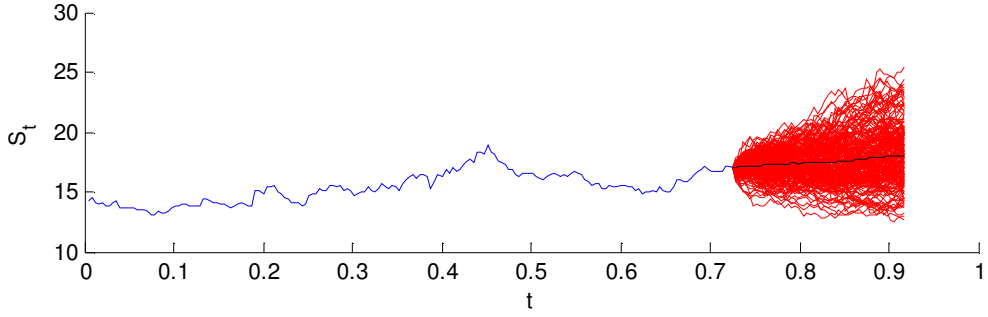
2.3.1 pav. Yahoo! Inc. istorinės akcijų kainos

Metodo kalibravimui pasirinktos Yahoo! Inc. (YHOO) istorinės akcijų kainos nuo 2010 01 04 iki 2010 09 27 [7]. Istorinės akcijų kainos pavaizduotos 2.3.1 paveiksle.

Atlikus Kolmogorovo-Smirnovo testą akcijų grąžų logaritmams, gauta, kad $p = 0.992$ ir $D = 0.0742$. Įvertinus tai, kad $D < p$ galima teigti, kad grąžų logaritmai pasiskirstę pagal normalųjį skirstinį, o duomenys yra tinkami klasikiniam akcijų kainų dinamikos modeliui.



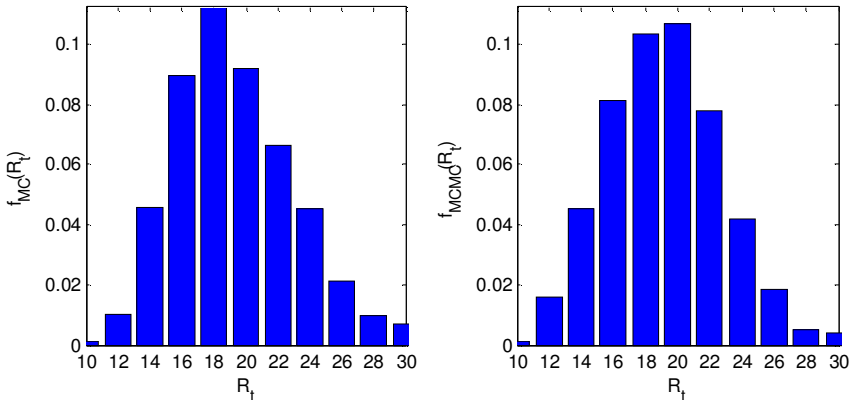
a)



b)

2.3.2 pav. Klasikinio Monte Carlo (*a*) modeliavimo ir MCMC (*b*) palyginimas

Kiekviena technika buvo sumodeliuota po 200 trajektorijų (2.3.2 pav.). Klasikiniu Monte Karlo metodu sumodeliuota vidutinė akcijos kaina po 50 dienų gauta 18.08 $. Naujai pasiūlytos technikos rezultatas yra 18.10 $ už vieną akciją. Panašios reikšmės buvo galima tikėtis, nes buvo įvertinta slinktis (trendas).

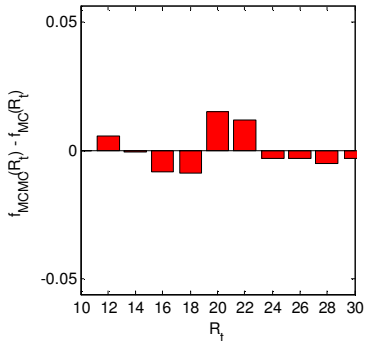


a ***b***

2.3.3 pav. Monte Karlo (*a*) ir MCMC (*b*) rezultatų palyginimas

2.3.3 paveiksle palygintos klasikinio Monte Karlo ir MCMC metodu sumodeliuotų akcijų kainų histogramos. Jos nusako modeliavimo pabaigoje gautų akcijų kainų pasiskirstymą.

Modeliavimo proceso metu sugeneruota 1000 akcijos kainų trajektorijų ir sumodeliuotos 100 dienų. Todėl reikėjo sugeneruoti 100000 atsitiktinių akcijų kainų grąžų.



2.3.4 pav. Skirtumai tarp Monte Karlo ir MCMC modeliavimų rezultatų

Didžiausi skirtumai tarp histogramų yra vidurkio srityje. Pasiskirstymo uodegos sutampa labiau. Didinant trajektorijų skaičių, klasikinis Monte Karlo metodas konverguoja į akcijų kainų pasiskirstymą; šiame darbe pasiūlytas metodas turi elgtis taip pat. Naujojo metodo sulyginimas su Monte Karlo metodu yra ekvivalentus tikrinimui, ar šis metodas konverguoja į akcijos kainos pasiskirstymą. Dažnai įvertinant skirtumą tarp dviejų tikimybinių pasiskirstymų yra naudojamas jų skirtumo modulio integralas. Nagrinėkime tokį įvertį:

$$\Delta_{MC} = \sum_{j=1}^{m} \left| h_{MC}\left(x_j\right) - h_{MCMC}\left(x_j\right) \right|, \qquad \textbf{(2.4.1)}$$

$h_{MC}(\cdot)$ ir $h_{MCMC}(\cdot)$ yra modeliavimo pabaigoje gautų akcijų kainų histogramos, m yra stulpelių skaičius, o x_j žymi j-tojo stulpelio centrą.

2.3.1 lentelė

Skirtumai tarp akcijų kainų histogramų, gautų Monte Karlo ir MCMC metodais

Stulpelių $g(x)$ skaičius	Trajektorijų skaičius	Atsitiktinių skaičių kiekis	Δ_{MC}
3	50	5000	0.245
	100	10000	0.131
	200	20000	0.116
	500	50000	0.072
	1000	100000	0.064

2.3.1 lentelė parodo kaip Δ_{MC} keičiasi, kai akcijų kainų trajektorijų skaičius N didėja. Kuo didesnis N tuo labiau Monte Karlo ir MCMC rezultatai sutampa. Eksperimentiniu būdu įrodyta, kad MCMC tinka modeliuoti akcijų kainas.

Stulpelių skaičiaus $g(x)$ parinkimas priklauso nuo pikų ir atstumų tarp jų tiksliniame skirstinyje. Kadangi akcijų kainų grąžų pasiskirstymo tankio funkcijos forma panaši į normalųjį skirstinį, galima spėti, kad $g(x)$ turėtų būti sudarytas iš nedidelio nelyginio skaičiaus stulpelių, kad būtų kuo panašesnis į tikslinį tankį.

2.3.2 lentelė

Skirtumai tarp akcijų kainų histogramų, gautų Monte Karlo ir MCMC metodais

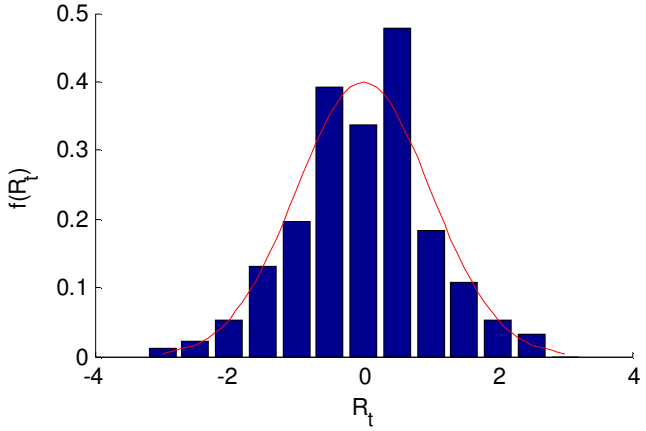
Stulpelių $g(x)$ skaičius	Trajektorijų skaičius	Atsitiktinių skaičių kiekis	Δ_{MC}
5	50	5000	0.265
	100	10000	0.131

	200	20000	0.091
	500	50000	0.060
	1000	100000	0.050

Iš 2.3.2 lentelės matyti, kad sudarant tikslinį tankį iš 5 stulpelių, gaunamas tikslesnis akcijų kainų pasiskirstymas. Didinant tikslumą ilgėja skaičiavimų laikas. Taip yra dėl to, kad reikia daugiau skaičiavimų norint surasti kuriam iš $g(x)$ intervalų priklauso tam tikras atsitiktinis dydis.

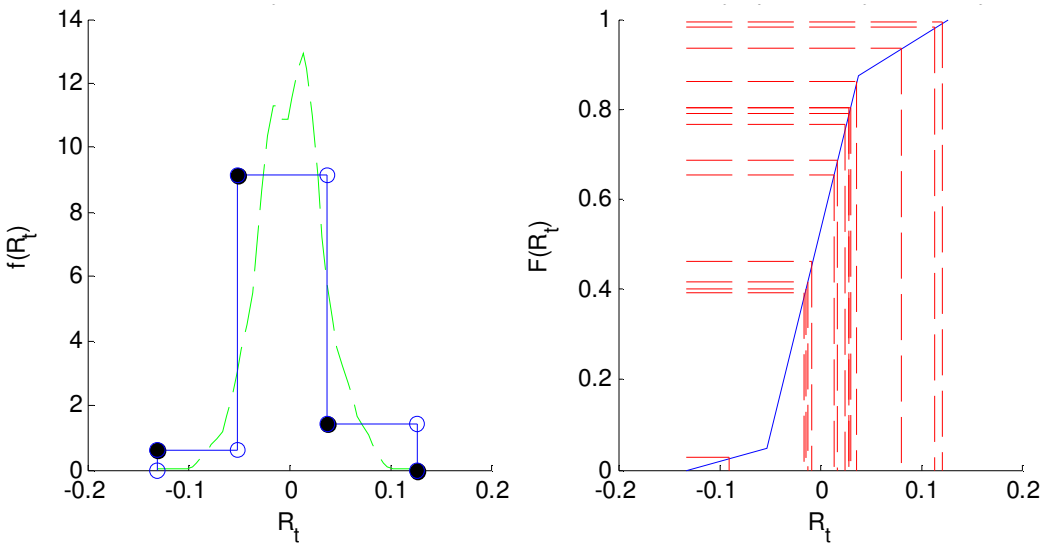
2.3.4. PASIRINKIMO SANDORIO ĮKAINOJIMO MCMC METODU PAVYZDYS

Toliau pateiktas pavyzdys, kai klasikinis Monte Karlo metodas negali būti taikomas modeliuoti akcijų kainoms.



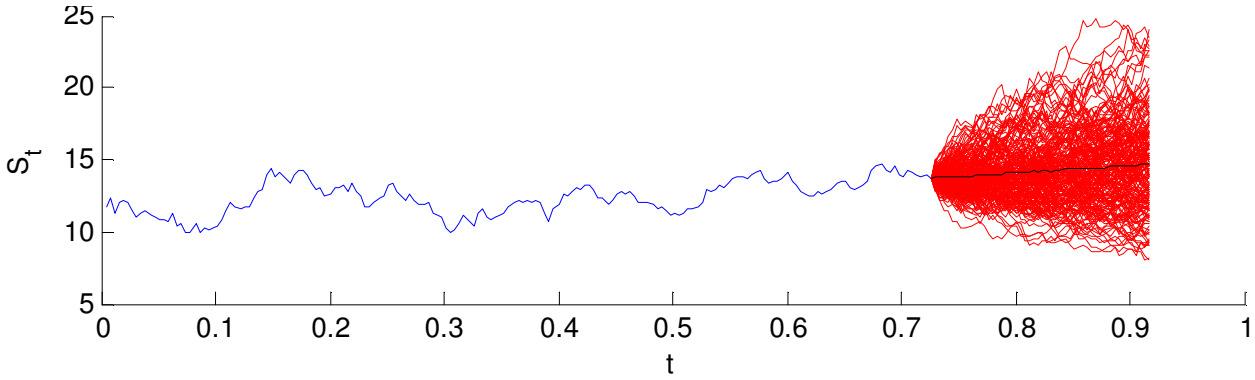
2.3.5 pav. Normalizuotų dienos grąžų logaritmų pasiskirstymas

Normalizuotų tolygiųjų dienos grąžų R_i logaritmų histograma kompanijos Tesco Corporation (TESO) akcijoms pavaizduota 2.3.5 paveiksle. Istoriniai duomenys paimti iš [7]. Nors normalumo hipotezė yra priimta, bet histogramoje yra 2 pikai. Jeigu esame užtikrinti, kad histograma visiškai atspindi tikrąjį pasiskirstymą, tai prielaida apie normalumą turi būti atmesta ir standartinis Monte Karlo metodas negali būti taikomas.



2.3.6 pav. Akcijų grąžų empirinis ir alternatyvus pasiskirstymai

Konstruojant branduolinį tankio įvertį grąžoms $r_i = \frac{S_i - S_{i-1}}{S_{i-1}}$ ir naudojant Gauso branduolio funkciją taip pat gaunamas 2 pikus turintis pasiskirstymas (2.3.6 pav.). Šiai tankio funkcijai buvo pritaikytas MCMC metodas. Kaip alternatyvus skirstinys naudotas dalimis tolygusis pasiskirstymas.



2.3.7 pav. TSO akcijų kainų prognozavimas

Vidutinė akcijos kaina po 50 dienų gauta lygi 14.75 $. Visos generuojamos akcijų kainų grąžos yra pasiskirstę pagal tikslinio tankio branduolinį tankio įvertinį. Modeliavimas yra visiškai paremtas empiriniais duomenimis ir neturi jokių prielaidų apie tikslinio tankio pasiskirstymą.

2.3.3 lentelė

TSO pasirinkimo pirkti sandorio kainos

Esamoji akcijos kaina	Sandorio įvykdymo kaina	Palūkanų norma	Tiksli kaina (Black-Scholes)	Sumodeliuota MCMC kaina	Santykinė paklaida
11,81	8,50	0,05	3,336	3,240	-0,029
	9,00		2,841	2,747	-0,033
	9,50		2,354	2,263	-0,039
	10,00		1,886	1,801	-0,045
	10,50		1,452	1,377	-0,052

2.3.3 lentelėje pateiktos pasirinkimo pirkti sandorio kainos, gautos iš 5000 MCMC metodu sumodeliuotų akcijų trajektorijų, kai sandorio trukmė yra 15 dienų. Visa sandorio trukmė buvo padalinta į 30 periodų. MCMC metodu gaunamos keliais procentais mažesnės negu apskaičiuotos pagal Black-Scholes formulę pasirinkimo sandorių kainos. Tai galima paaiškinti žvelgiant į 2.3.5 paveikslą, artimų 0 grąžų yra daugiau negu jų būtų normalumo prielaidą tenkinančiame pasiskirstyme. Modeliavimo metu tai lėmė daugiau mažesnių kainų pokyčių.

3. PROGRAMINĖ REALIZACIJA IR INSTRUKCIJA VARTOTOJUI

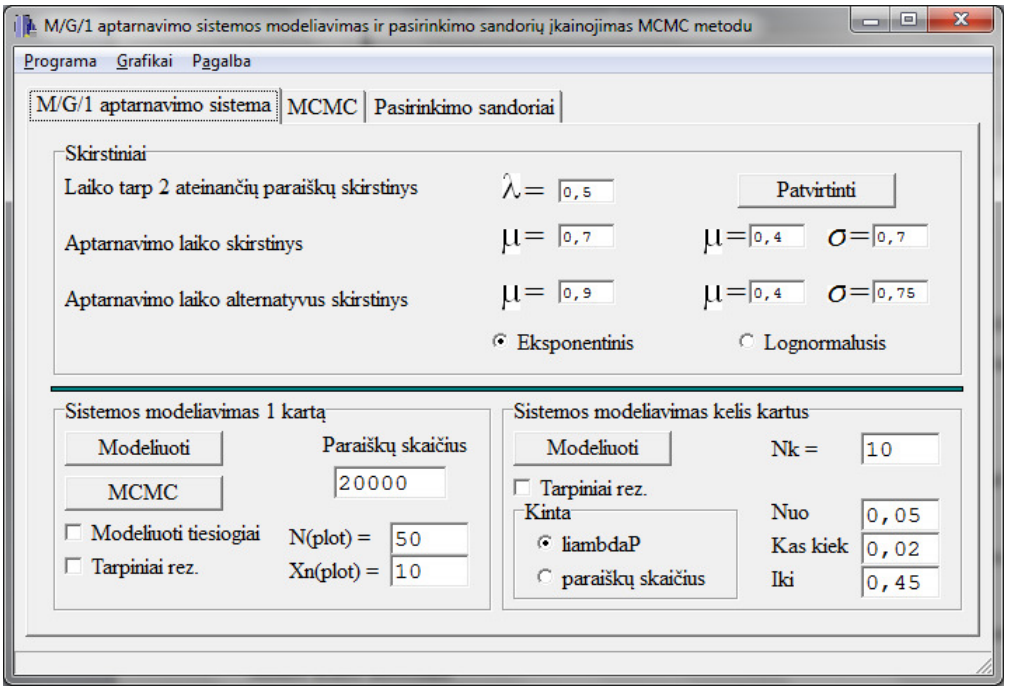
3.1. PROGRAMINĖS REALIZACIJOS PASTABOS

Tyrimui programa sukurta su C++ Builder 5 programavimo aplinka. C++ kalba pasirinkta todėl, kad greitai atlieka skaičiavimus ir turi standartinius komponentus, kurių pakanka pagrindinius rezultatus atvaizduoti grafiškai.

Yra nemažai specializuotų programinių priemonių finansiniams skaičiavimams. Pavyzdžiui MS Excel yra labai patogi skaičiuoklė, tačiau jos naudojama vidinė programavimo kalba VBA veikia lėtai.

Standartinis atsitiktinių skaičių generatorius įvairiose programavimo kalbose dažnai būna neefektyvus greičio arba atsitiktinių skaičių kokybės atžvilgiu. Įvertinus tai buvo panaudotas Park-miller atsitiktinių skaičių generatorius, kuris yra kokybiškesnis už standartinę C++ funkciją *rand()* [9].

3.1. PROGRAMOS INSTRUKCIJA VARTOTOJUI

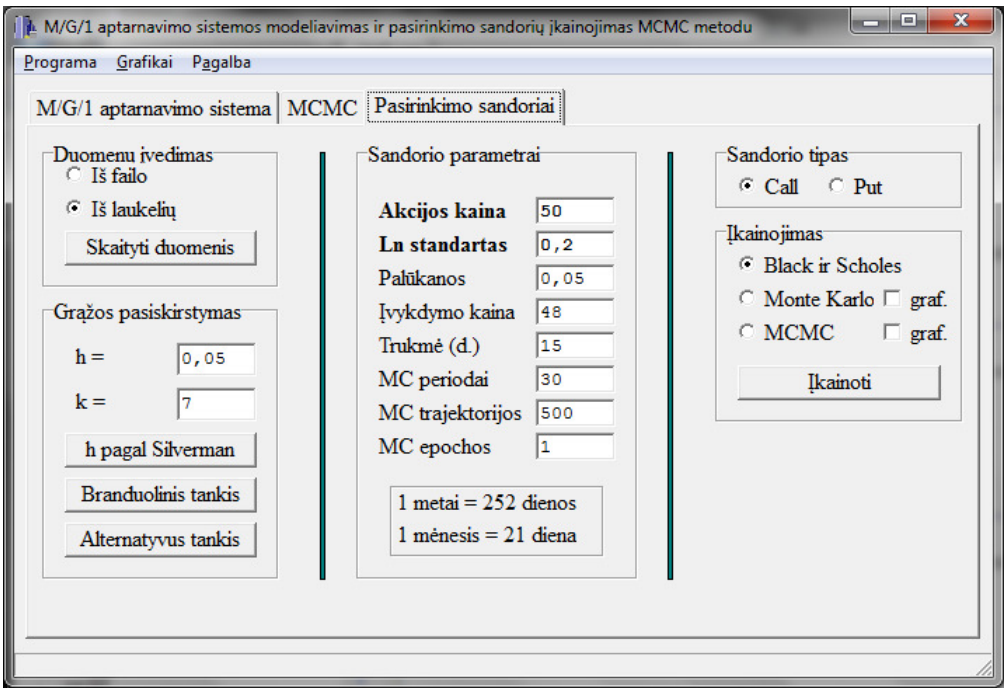


3.1.1 pav. Pagrindinis programos langas (aptarnavimo sistema)

Modeliuojant aptarnavimo sistemą, reikia pasirinkti paraiškos aptarnavimo laiko pasiskirstymo tipą: eksponentinį arba lognormalųjį (3.1.1 pav.). Programa priima, kad nors skirstinys yra pasirinktas, bet jis yra nežinomas. Laiko tarp dviejų ateinančių paraiškų skirstinys visada yra eksponentinis. Kiekvienam iš išvardintų tikimybinių dėsnių atitinkamuose laukeliuose įvedami reikiami parametrai.

Nežinomam skirstiniui pasirenkame alternatyvų įvesdami jo parametrus į atitinkamus laukelius. Aptarnavimo sistema gali būti modeliuojama tik 1 kartą arba po daug kartų kintant paraiškų atėjimo srautui arba modeliuojamų paraiškų kiekiui.

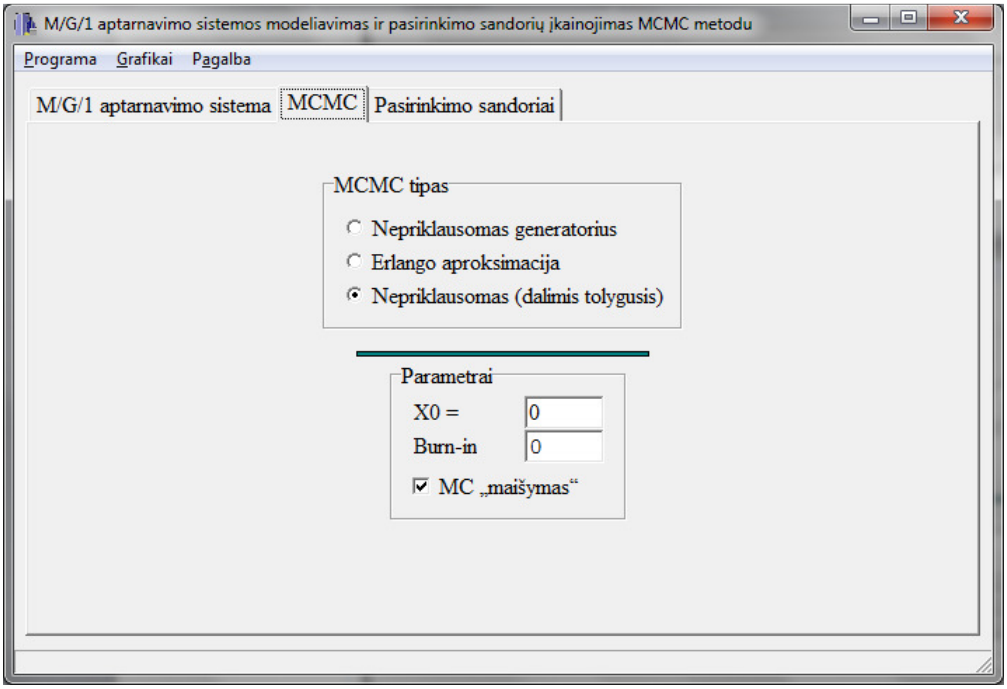
Taip pat galima tik sugeneruoti (mygtukas „MCMC“) paraiškų aptarnavimo momentus ir taip patyrinėti MCMC konvergavimą ir rezultatus.



3.1.2 pav. Pagrindinis programos langas (pasirinkimo sandoriai)

Modeliuojant akcijų kainas ir tokiu būdu įkainojant pasirinkimo sandorius, įvedami istoriniai duomenys iš pasirinkto duomenų failo arba naudojama įvesta esamoji akcijos kaina ir jos grąžos logaritmo standartas (3.1.2 pav.). Tokiu atveju pasirinkimo sandoris nagrinėjamai akcijai gali būti įkainojamas tik pagal Black-Scholes arba Monte Karlo metodais.

Pasirinkimo sandorių kaina apskaičiuojama pagal Black-Scholes formulę arba modeliuoti MCMC ar Monte Karlo metodais.



3.1.3 pav. Programos langas (MCMC parametrai)

Norint patyrinėti akcijos kainos dinamiką keičiant grąžų skirstinio branduolio plotį, užtenka norimą jo reikšmę įrašyti į atitinkamą laukelį.

Meniu punkte „Grafikai“ galima pasirinkti iš daugelio variantų ką pavaizduoti. Jei prašoma vaizduoti, pavyzdžiui, branduolinį tankį, o jis dar nesudarytas, tai programa parodys atitinkamą pranešimą.

Meniu punkte „Pagalba“ pateikiama trumpa informacija apie programą, ja modeliuojamas stochastines sistemas.

Bet kurį grafiką galima išsaugoti vektoriniu *wmf* formatu. Saugoma paskutiniame aktyviame kataloge. Rezultatų langą taip pat galima išsaugoti į rezultatų failą.

Sukurtą programinę įrangą galima toliau tobulinti ir naudoti kaip bazę kuriant G/G/1 eilių sistemos modelį arba įkainoti taip vadinamus egzotinius (sudėtingus) pasirinkimo sandorius.

4. DISKUSIJA

4.1. SUDĖTINGŲ SKIRSTINIŲ MODELIAVIMAS BENDRU ATVEJU

Šiame skyrelyje bus pateiktas algoritmas, kuris nurodo, kaip darbo metu tirtą sudėtingų skirstinių modeliavimo metodiką taikyti bendru atveju. Ši diskusija yra aktuali, nes dažnai realiai stochastinei sistemai nėra universalių modeliavimo metodų. Dėl įvairių prielaidų apie turimus duomenis dažnai turimi metodai taikomi neadekvačiai.

Tarkime, kad tiriama reali stochastinė sistema, reikia sumodeliuoti kurį nors jos parametrą.

Empirinio parametro pasiskirstymo įvertinimas. Nustatoma, koks yra dominančio parametro pasiskirstymas. Jeigu žinoma tikimybinio tankio funkcijos išraiška, ji priimama kaip tikslinis skirstinys, kurį reikia modeliuoti.

Sistemos modeliavimas 1. Kai tankio funkcijos išraiška paprasta, taikomas tiesioginis modeliavimas (atvirkštinė skirstinio funkcija). Išraiškai esant per daug sudėtingai galima taikyti atsitiktinio dydžio aproksimavimą eksponentinių skirstinių mišiniu, Erlango, Normaliojo ar Gama skirstiniais. Ir taip pat taikomas tiesioginis modeliavimas.

Sistemos modeliavimas 2. Kai nežinoma nei kaip aproksimuoti, nei analizinės modeliuojamo parametro tikimybinio tankio funkcijos, galima taikyti darbe nagrinėtą metodiką. Pirmiausiai įvertinamas tikslinis tankis naudjant branduolinį tankio įvertį. Tada sukonstruojamas jam alternatyvus (dalimis tolygusis) tankis, kurį modeliuoti nesunku. Taikomas MCMC metodas ir sumodeliuojamas sudėtingą pasiskirstymą turintis parametras.

IŠVADOS

1. MCMC metodu sugeneruotų atsitiktinių skaičių sekos tiesiogiai naudoti sitemų modeliavimui negalima, nes gautoji seka yra priklausoma. Atsitiktinis sekos sumaišymas yra būtinas, norint gauti nepriklausomų atsitiktinių dydžių seką. Tai ženkliai pagerina modeliavimo rezultatus.

2. Eksperimentiškai nustatyta, kad išrenkant iš sekos kas *k*-ąjį narį ($k > 4$) gaunama atsitiktinių dydžių seka, kokybiškesnė už pradinę (nemaišytą).

3. Nustatyta aptarnavimo sistemos charakteristikų paklaidų empirinė priklausomybė nuo santykio μ/λ.

4. Akcijos grąžų tikimybiniam pasiskirstymui naudojamas branduolinio tankio įvertis leidžia gauti reprezentatyvų ir glotnų tankio įvertį bet kokio dydžio imčiai. Toks tankio įvertis vertina ir išskirtis (jei jos nešalinamos). Tokiu būdu galima modeliuoti rinką be arba su ekstremaliais įvykiais.

5. Iš modeliavimo rezultatų matyti, kad pasiūlyta akcijų kainų modeliavimo technika generuoja mažą dispersiją turinčią vidutinę akcijų kainų trajektoriją, o dispersija didėja laike.

6. Akcijos dinamikos modeliavimo rezultatų tikslumas didėja, didinant dalimis tolygiųjų pasiskirstymų, iš kurių sudarytas alternatyvus skirstinys, skaičių.

7. MCMC metodas ir branduolinis tankio įvertis leidžia modeliuoti bet kokią realią sistemą. Tikslinis tankis yra konstruojamas remiantis statistine informacija.

8. Atlikus tyrimus nustatyta, kad geriausios kokybės atsitiktiniai skaičiai gaunami MCMC metodu generuojant ne mažiau 100000 elementų imtį, ir/arba alternatyvų skirstinį formuojant iš didesnio skaičiaus tolygiųjų pasiskirstymų.

9. Tyrimas parodė, kad sukurti modeliai yra adekvatūs teoriniams modeliams.

LITERATŪRA

1. Bolch, G.; Greiner, S.; de Meer, H.; Trivedi, K., S. Queuing networks and Markov Chains. – John Wiley & Sons, 2006. – 878p.

2. Dagpunar J. S., Simulation and Monte Carlo: With applications in finance and MCMC. John Wiley & Sons, Ltd, 2007. 348p.

3. Eraker B. MCMC Analysis of diffusion models with applications to finance. Journal of Business and Economic Statistics, vol. 19, pp. 177-191, 2001.

4. Fischer B., Myron S. The Pricing of Options and Corporate Liabilities // The Journal of Political Economy , Vol. 81, No. 3. – 1973. 637-659 p.

5. Hastings W. K., Monte Carlo Sampling Methods Using Markov Chains and Their Applications // Biometrika, Vol. 57, No. 1.- 1970, p. 97-109.

6. Hull J. C. Options, Futures, and Other Derivatives, 6ed. Prentice Hall, 2006. 816 p.

7. Yahoo! Finance [žiūrėta 2011-05-19]. Prieiga per internetą: http://finance.yahoo.com/

8. Johannes M., Polson N. MCMC Methods for Continuous-Time Financial Econometrics, 2006 [žiūrėta 2011-05-19]. Prieiga per internetą: http://www0.gsb.columbia.edu/faculty/mjohannes/PDFpapers/JP_2006.pdf

9. Joshi M. S. C++ Design Patterns and Derivatives Pricing, 2ed. New York, Cambridge University Press, 2008. 292 p.

10. Paranevičius H., Valakevičius E. Markovo procesų teorijos taikymas sistemoms modeliuoti. - Kaunas: Technologija, 1991. 91p.

11. Prokaj V., Proposal selection for MCMC simulation // Applied Stochastic Models and Data Analysis .- ISBN 978-9955-28-463-5 .- 2009, p. 61–65.

12. Song-Chun Z., Dellcert F., Zhouwen T., Markov Chain Monte Carlo for Computer Vision // 10 Conf. on Computer Vision. 2005.

13. Silverman B. W. Kernel Density Estimation Using the Fast Fourier Transform // Journal of the Royal Statistical Society. Series C (Applied Statistics), Vol. 31, No. 1 – 1982. 93-99 p.

14. Tijms H. C. A First Course in Stochastic Models. Great Britain, Padstow, Cornwall, T J International ltd, 2003. 488 p.

15. Valakevičius E. Investavimas finansų rinkose. Kaunas, Technologija, 2008. 340 p.

16. Wilmott P. Introduces Quantitative Finance, 2ed. John Willey & Sons Ltd., 2007. 722 p.

PUBLIKACIJOS IR PRANEŠIMAI KONFERENCIJOSE

1. M. Landauskas, E. Valakevičius – Sumodeliuotų *M/G/1* eilių sistemos charakteristikų tikslumo priklausomybė nuo jos parametrų („Taikomoji matematika“, 2010-04-30).
2. M. Landauskas, E. Valakevičius – MCMC Approach to Modelling Queuing Systems (Lietuvos matematikų draugijos 51 – joji konferencijoja, 2010-06-17).

PRANEŠIMAI KONFERENCIJOSE

1. M. Landauskas, E. Valakevičius – Aptarnavimo sistemų modeliavimas MCMC metodu („Matematika ir matematikos dėstymas“, 2010-04-09).
2. M. Landauskas, E. Valakevičius – Akcijų kainų modeliavimas MCMC metodu („Operacijų tyrimai verslui ir socialiniams procesams“, 2010-10-01). (*straipsnis recenzuojamas*).
3. M. Landauskas, E. Valakevičius – Universalus alternatyvaus skirstinio MCMC metodui parinkimas („Matematika ir matematikos dėstymas“, 2011-04-08).

1 PRIEDAS. PROGRAMOS C++ TEKSTAI

Unit1.h

```
//---------------------------------------------------------------------------
#ifndef Unit1H
#define Unit1H
#include <Classes.hpp>
#include <ComCtrls.hpp>
#include <Controls.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
#include <Menus.hpp>
#include <StdCtrls.hpp>
#include <Graphics.hpp>
//---------------------------------------------------------------------------
// Naudojant random_shuffle() reikia atskiros rand() funkcijos.
//---------------------------------------------------------------------------
#include <algorithm>  // random_shuffle()
ptrdiff_t myrandom(ptrdiff_t i)
{
             return rand()%i;
}
// Rodyklė į atskirą rand() funkciją
ptrdiff_t (*p_myrandom)(ptrdiff_t) = myrandom;
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Chart.hpp>
#include <ExtCtrls.hpp>
#include <TeEngine.hpp>
#include <TeeProcs.hpp>
#include <Series.hpp>
#include <Graphics.hpp>
#include <Menus.hpp>
#include <ComCtrls.hpp>
#include <Dialogs.hpp>
//---------------------------------------------------------------------------
#include <fstream>
#include <iomanip>
using namespace std;
//---------------------------------------------------------------------------
#include "MCMC.h"
#include "Eile.h"
#include "Apie.h"
#include "Grafikas.h"
#include "Modeliavimas.h"
#include "MCarlo.h"
#include <time.h>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:  // IDE-managed Components
             TMainMenu *MainMenu1;
             TMenuItem *Programa1;
             TMenuItem *Ieiti1;
             TMenuItem *Grafikai1;
             TMenuItem *ppxPLOT;
             TMenuItem *paxPLOT;
             TMenuItem *pqxPLOT;
             TMenuItem *FaxPLOT;
             TMenuItem *FqxPLOT;
             TMenuItem *N2;
             TMenuItem *teilPLOT;
             TMenuItem *taptPLOT;
             TMenuItem *N3;
             TMenuItem *savePLOT;
             TMenuItem *pixPLOT;
             TMenuItem *pixPROCESS;
             TMenuItem *delPLOTS;
             TMenuItem *ErlPLOT;
             TMenuItem *stagePLOT;
             TMenuItem *Epaxqax1;
             TMenuItem *Epaxx1;
             TMenuItem *Epaxerlx1;
             TMenuItem *Pagalba1;
             TMenuItem *Apie;
```

```
            TMenuItem *EPEilPaklaidos;
            TMenuItem *EPSistPaklaidos;
            TMenuItem *ETEilPaklaidos;
            TMenuItem *ETSistPaklaidos;
            TStatusBar *sb;
    TPageControl *PageControl1;
    TTabSheet *TabSheet1;
    TTabSheet *TabSheet2;
    TGroupBox *GroupBox1;
    TLabel *Label6;
    TLabel *Label7;
    TLabel *Label9;
    TLabel *Label11;
    TEdit *Edit3;
    TButton *Button11;
    TEdit *Edit5;
    TEdit *Edit4;
    TRadioButton *RadioButton3;
    TRadioButton *RadioButton4;
    TEdit *Edit9;
    TEdit *Edit10;
    TEdit *Edit11;
    TEdit *Edit12;
    TGroupBox *GroupBox2;
    TLabel *Label4;
    TButton *Button7;
    TCheckBox *NoMCMC;
    TButton *Button3;
    TEdit *Edit2;
    TCheckBox *CheckBox1;
    TGroupBox *GroupBox4;
    TLabel *Label18;
    TLabel *Label3;
    TLabel *Label14;
    TLabel *Label19;
    TButton *Button6;
    TEdit *Edit14;
    TRadioGroup *RadioGroup2;
    TRadioButton *RadioButton5;
    TRadioButton *RadioButton6;
    TEdit *Edit15;
    TEdit *Edit16;
    TEdit *Edit17;
    TCheckBox *CheckBox2;
    TRadioGroup *RadioGroup1;
    TTabSheet *TabSheet3;
    TOpenDialog *OpenDialog1;
    TGroupBox *GroupBox5;
    TRadioButton *RadioButton7;
    TRadioButton *RadioButton8;
    TButton *Button1;
    TGroupBox *GroupBox9;
    TLabel *Label23;
    TLabel *Label24;
    TLabel *Label25;
    TLabel *Label30;
    TLabel *Label31;
    TLabel *Label32;
    TLabel *Label33;
    TLabel *Label34;
    TEdit *Edit18;
    TEdit *Edit19;
    TEdit *Edit20;
    TEdit *Edit30;
    TEdit *Edit34;
    TEdit *Edit38;
    TEdit *Edit39;
    TEdit *Edit40;
    TShape *Shape3;
    TShape *Shape4;
    TGroupBox *GroupBox12;
    TButton *Button13;
    TRadioButton *RadioButton24;
    TRadioButton *RadioButton25;
    TCheckBox *CheckBox5;
    TGroupBox *GroupBox14;
    TLabel *Label35;
    TLabel *Label36;
    TMenuItem *Rodytilegend1;
    TMenuItem *Rodytirezultatus1;
```

```
    TMenuItem *Saugotirezultatus1;
    TMenuItem *N5;
    TMenuItem *N6;
    TMenuItem *Rodytigrafik1;
    TRadioButton *RadioButton9;
    TCheckBox *CheckBox6;
    TGroupBox *GroupBox7;
    TLabel *Label13;
    TLabel *Label17;
    TEdit *Edit6;
    TEdit *Edit8;
    TCheckBox *Shuffle;
    TEdit *Edit13;
    TLabel *Label16;
    TMenuItem *Groslogaritmobranduolinistankis1;
    TMenuItem *Groslogaritmohistograma1;
    TMenuItem *Paklaidos2;
    TMenuItem *Pasiskirstymofunkcijos1;
    TMenuItem *Tankiofunkcijos1;
    TMenuItem *Akcijos1;
    TMenuItem *N1;
    TMenuItem *Istorinkaina1;
    TMenuItem *Alternatyvausskhistograma1;
    TMenuItem *Teorinisgrostankis1;
    TGroupBox *GroupBox13;
    TRadioButton *RadioButton18;
    TRadioButton *RadioButton19;
    TImage *Image1;
    TImage *Image2;
    TImage *Image3;
    TLabel *Label1;
    TLabel *Label2;
    TMenuItem *Alternatyviskirstiniofunkcija1;
    TGroupBox *GroupBox6;
    TButton *Button4;
    TEdit *Edit48;
    TButton *Button8;
    TEdit *Edit47;
    TLabel *Label8;
    TLabel *Label10;
    TButton *Button2;
    TImage *Image4;
    TLabel *Label12;
    TImage *Image5;
    TLabel *Label26;
    TImage *Image6;
    TLabel *Label27;
    TImage *Image7;
    TLabel *Label28;
    TMenuItem *MG1sistema1;
    TMenuItem *N7;
    TEdit *Edit1;
    TLabel *Label5;
    TRadioButton *MCMCtype1;
    TRadioButton *MCMCtype3;
    TRadioButton *MCMCtype4;
    TShape *Shape1;
    TShape *Shape2;
              void __fastcall Form1Close(TObject *Sender, TCloseAction &Action);
              void __fastcall Button11Click(TObject *Sender);
    void __fastcall Button6Click(TObject *Sender);
              void __fastcall Button7Click(TObject *Sender);
              void __fastcall ppxPLOTClick(TObject *Sender);
              void __fastcall paxPLOTClick(TObject *Sender);
              void __fastcall pqxPLOTClick(TObject *Sender);
              void __fastcall FaxPLOTClick(TObject *Sender);
              void __fastcall FqxPLOTClick(TObject *Sender);
              void __fastcall teilPLOTClick(TObject *Sender);
              void __fastcall taptPLOTClick(TObject *Sender);
              void __fastcall Ieiti1Click(TObject *Sender);
              void __fastcall savePLOTClick(TObject *Sender);
              void __fastcall pixPLOTClick(TObject *Sender);
              void __fastcall pixPROCESSClick(TObject *Sender);
              void __fastcall delPLOTSClick(TObject *Sender);
              void __fastcall ErlPLOTClick(TObject *Sender);
              void __fastcall stagePLOTClick(TObject *Sender);
              void __fastcall Epaxqax1Click(TObject *Sender);
              void __fastcall Epaxx1Click(TObject *Sender);
              void __fastcall Epaxerlx1Click(TObject *Sender);
              void __fastcall ApieClick(TObject *Sender);
```

```
            void __fastcall Form1Create(TObject *Sender);
            void __fastcall EPEilPaklaidosClick(TObject *Sender);
            void __fastcall EPSistPaklaidosClick(TObject *Sender);
            void __fastcall ETEilPaklaidosClick(TObject *Sender);
            void __fastcall ETSistPaklaidosClick(TObject *Sender);
   // Included from Bachelors software
   void __fastcall Button1Click(TObject *Sender);
   void __fastcall Button13Click(TObject *Sender);
   void __fastcall Button10Click(TObject *Sender);
   void __fastcall RadioButton18Click(TObject *Sender);
   void __fastcall RadioButton19Click(TObject *Sender);
   void __fastcall Button4Click(TObject *Sender);
   void __fastcall Rodytilegend1Click(TObject *Sender);
   void __fastcall Rodytirezultatus1Click(TObject *Sender);
   void __fastcall Saugotirezultatus1Click(TObject *Sender);
   void __fastcall Rodytigrafik1Click(TObject *Sender);
   void __fastcall Button8Click(TObject *Sender);
   void __fastcall Button3Click(TObject *Sender);
   void __fastcall Groslogaritmohistograma1Click(TObject *Sender);
   void __fastcall Groslogaritmobranduolinistankis1Click(TObject *Sender);
   void __fastcall Istorinkaina1Click(TObject *Sender);
   void __fastcall Alternatyvausskhistograma1Click(TObject *Sender);
   void __fastcall Teorinisgrostankis1Click(TObject *Sender);
   void __fastcall Alternatyviskirstiniofunkcija1Click(TObject *Sender);
   void __fastcall Button2Click(TObject *Sender);
   void __fastcall RadioButton5Click(TObject *Sender);
   void __fastcall RadioButton6Click(TObject *Sender);
private:      // User declarations
            MCMC Sampler;
            int N, eilesN;
   int maxBusena;
   int isViso;
   double T, pabaiga;
   double liambdaP, liambdaA, liambdaQ;
            double * X;
   double * Prop;  // Sugeneruotam alternatyviam skirstiniui saugoti
   TEile * Eile;
   double * Busenos;
   double ** rezultatai;
   void NaikintiEile(TEile * trash);
   void RodytiMasyva(int n, int * mas1, double * mas2, AnsiString comment);
   void RodytiEilesPokycius(TEile * obj, AnsiString comment);
   void SkaitytiParametrus();
   void SkaitytiParametrus2(double param);
   void MasyvasMCMC();
   void ModeliuotiSistema();
   double Momentas(double * M, int n, int k);
   double * TeoriniaiSistemosParametrai(double * parametrai = NULL);
   double * EmpiriniaiSistemosParametrai(double * parametrai = NULL);
   void PalygintiParametrus(double * teoriniai, double * empiriniai);
   void RodytiMatrica(double ** M, int eil);
   void RodytiRezultatus(double ** M, int eil);
   void VidutiniuPaklaiduPriklausomybe(double ** rezultatai, int id,
                                       double nuo, double kas, int n);
   double KTauAlfa(double tau, double alfa, double beta);
   void RastiAlfa(double * X, double * Y, int N, double & A, double & B);
   Modelis M;
   MCarlo MC;
   __fastcall TForm6(TComponent* Owner);
public:                    // User declarations
            __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

Unit1.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "math.h"
#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
```

```
              : TForm(Owner)
{
}
//---------------------------------------------------------------------------
// Vykdomas tik MCMC metodas.
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
              MasyvasMCMC();
   // Sumodeliuota imtis išvedama į duomenų failą
   ofstream fr(("MCMC sample " + IntToStr(N) + ".txt").c_str());
   for (int i = 0; i < N; i++){
              fr << X[i];
      if (i < N - 1)
              fr << endl;
   }
   Form5->Show();
}
//---------------------------------------------------------------------------
// Aptarnavimo sistemos laikų generavimas MCMC metodu arba tiesiogiai.
//---------------------------------------------------------------------------

void TForm1::MasyvasMCMC()
{
   N = StrToInt(Edit1->Text);
   double X0 = StrToFloat(Edit8->Text);

   // MCMC tipas

   if (MCMCtype1->Checked){
              Sampler.MCMCtype = "Independence sampler";
   }
   /*
   else if (MCMCtype4->Checked){
              Sampler.MCMCtype = "Independence piecewise";
   }
   */
   else{
      MCMCtype3->Checked = true;
              Sampler.MCMCtype = "Erlang approximation";
   }

   // Modeliuojamo skirstinio tipas

   if (RadioButton3->Checked){
              Sampler.DISTtype = "exp";
   }
   else if (RadioButton4->Checked){
              Sampler.DISTtype = "lognorm";
   }

   Sampler.SetX0(X0);
   int burn_in = StrToInt(Edit6->Text);
   Sampler.Burn_in(burn_in);

   if (MCMCtype3->Checked)
              Sampler.InitErlang(100000);

   int N = StrToInt(Edit1->Text);
   if (X) delete X;
   X = NULL;
   X = new double [N];
   for (int i = 0; i < N; i++)
              if (NoMCMC->Checked)
              X[i] = Sampler.invFProposal(-1, -1);
      else
              if (Shuffle->Checked){
                            Sampler.GetNumber();
              Sampler.GetNumber();
              Sampler.GetNumber();
              Sampler.GetNumber();
                                          X[i] = Sampler.GetNumber();
              }
              else
                            X[i] = Sampler.GetNumber();
}
//---------------------------------------------------------------------------
// Eilę reprezentuojančių objektų naikinimas.
//---------------------------------------------------------------------------
```

```
void TForm1::NaikintiEile(TEile * trash)
{
              while (trash){
              if (trash->kita){
              TEile * trash2 = trash;
         trash = trash->kita;
         delete trash2;
      }
      else{
              delete trash;
         trash = NULL;
      }
   }
}
//---------------------------------------------------------------------------
// Atminties atlaisvinimas prieš uždarant programą.
//---------------------------------------------------------------------------

void __fastcall TForm1::Form1Close(TObject *Sender, TCloseAction &Action)
{
              if (X) delete [] X;
   if (Eile) NaikintiEile(Eile);
   if (Busenos) delete [] Busenos;
   if (Prop){
      delete [] Prop;
      Prop = NULL;
   }
   if (rezultatai){
              for (int i = 0; i < isViso; i++){
                           delete [] rezultatai[i];
              }
              delete [] rezultatai;
      rezultatai = NULL;
   }
}
//---------------------------------------------------------------------------
// Nuskaitomi aptarnavimo sistemos parametrai. Naudinga, kai tik norima
// nubraižyti aptarnavimo ar atėjimo į sistemą srautų pasiskirstymus.
//---------------------------------------------------------------------------

void __fastcall TForm1::Button11Click(TObject *Sender)
{
              SkaitytiParametrus();
}
//---------------------------------------------------------------------------
// Skirstinių parametrų įvedimas.
//---------------------------------------------------------------------------

void TForm1::SkaitytiParametrus()
{
              // Modeliuojamo skirstinio tipas
   if (RadioButton3->Checked){
              Sampler.DISTtype = "exp";
   }
   else if (RadioButton4->Checked){
              Sampler.DISTtype = "lognorm";
   }

              liambdaP = StrToFloat(Edit3->Text);
   liambdaA = StrToFloat(Edit4->Text);
   liambdaQ = StrToFloat(Edit5->Text);
   Sampler.SetLambdas(liambdaP, liambdaA, liambdaQ);

              if (RadioButton4->Checked){
              double p1 = StrToFloat(Edit9->Text);
              double p2 = StrToFloat(Edit10->Text);
              double p3 = StrToFloat(Edit11->Text);
              double p4 = StrToFloat(Edit12->Text);
      Sampler.SetLN(p1, p2, p3, p4);
   }
}
//---------------------------------------------------------------------------
// Automatiniam sistemos parametrų skaitymui.
//---------------------------------------------------------------------------

void TForm1::SkaitytiParametrus2(double param)
{
              // Modeliuojamo skirstinio tipas
   if (RadioButton3->Checked){
              Sampler.DISTtype = "exp";
```

```
   }
   else if (RadioButton4->Checked){
             Sampler.DISTtype = "lognorm";
   }

   if (RadioButton5->Checked)
             liambdaP = param;
   else
      liambdaP = StrToFloat(Edit3->Text);

   liambdaA = StrToFloat(Edit4->Text);
   liambdaQ = StrToFloat(Edit5->Text);
   Sampler.SetLambdas(liambdaP, liambdaA, liambdaQ);

             if (RadioButton4->Checked){
             double p1 = StrToFloat(Edit9->Text);
             double p2 = StrToFloat(Edit10->Text);
             double p3 = StrToFloat(Edit11->Text);
             double p4 = StrToFloat(Edit12->Text);
      Sampler.SetLN(p1, p2, p3, p4);
   }
}
//---------------------------------------------------------------------------
// Funkcija skaičių masyvo rodymui rezultatų lange.
//---------------------------------------------------------------------------

void TForm1::RodytiMasyva(int n, int * mas1, double * mas2, AnsiString comment)
{
   if (Form5->mm){
             Form5->mm->Lines->Add("");
      Form5->mm->Lines->Add(comment + " (N = " + IntToStr(n) + ")");
      AnsiString line = "";
      if (mas1)
             for (int i = 0; i < n; i++)
             line += " " + FloatToStrF((double)mas1[i], ffFixed, 8, 3);
      if (mas2)
             for (int i = 0; i < n; i++)
             line += " " + FloatToStrF(mas2[i], ffFixed, 8, 3);
      Form5->mm->Lines->Add(line);
      Form5->Show();
   }
}
//---------------------------------------------------------------------------
// Eilės pokyčių rodymas.
//---------------------------------------------------------------------------

void TForm1::RodytiEilesPokycius(TEile * obj, AnsiString comment)
{
   if (Form5->mm){
             TEile * rod = obj;
      int n = 0;
             Form5->mm->Lines->Add("");
      AnsiString line1 = "Laikas:  ";
      AnsiString line2 = "Pokytis: ";
      while (rod){
         line1 += " " + FloatToStrF(rod->t, ffFixed, 8, 3);
         line2 += " " + FloatToStrF(rod->pokytis, ffFixed, 8, 3);
         rod = rod->kita;
         n++;
      }
      Form5->mm->Lines->Add(comment + " (n = " + IntToStr(n) + ")");
      Form5->mm->Lines->Add(line1);
      Form5->mm->Lines->Add(line2);
      Form5->Show();
   }
}
//---------------------------------------------------------------------------
// Paraiškų ir aptarnavimo srauto generavimas.
//---------------------------------------------------------------------------

void __fastcall TForm1::Button7Click(TObject *Sender)
{
             Sampler.SetMemo(Form5->mm);
   Form5->mm->Visible = false;

             SkaitytiParametrus();
   N = StrToInt(Edit1->Text);
             ModeliuotiSistema();
   PalygintiParametrus(TeoriniaiSistemosParametrai(NULL), EmpiriniaiSistemosParametrai(NULL));
```

```
   Form5->mm->Visible = true;
   Form5->Show();
}
//---------------------------------------------------------------------------
// Eilių sistemos modeliavimas.
//---------------------------------------------------------------------------

void TForm1::ModeliuotiSistema()
{
   // Aptarnavimo srauto generavimas (paraiškų aptarnavimo trukmės).

   if (NoMCMC->Checked){
            N = StrToInt(Edit1->Text);
            if (X) delete X;
            X = NULL;
            X = new double [N];
            for (int i = 0; i < N; i++){
            double u = (double)rand() / ((double)(RAND_MAX + 1));
                        X[i] = Sampler.invFTarget(u, liambdaA);
      }
   }
   else
      MasyvasMCMC();

   if (Eile) NaikintiEile(Eile);
   Eile = NULL; eilesN = 0;
   TEile * rodykle = NULL;  // Darbinė rodyklė 1 (esamoji).
   TEile * rodykle2 = NULL;  // Darbinė rodyklė 2 (ankstesnė).

   // Paraiškų srauto generavimas (momentai, kuriais jos atėjo).

   // Objektų grandinėlės formavimas (modeliavimas).
   for (int i = 0; i < N; i++){  // Buvo (i = 1; ...)
      // Paraiška atėjo į sistemą.
      if (Eile){
            double u = (double)rand() / ((double)(RAND_MAX + 1));
            double rnd = rodykle2->t + Sampler.invFExp(u, liambdaP);
            rodykle = rodykle->IterptiMomenta(rnd, 1);
      }
      else{
            Eile = new TEile(0, 1);
         rodykle = Eile;
      }
      rodykle->tAptarnavimo = X[i];
      if (i == 0)  // Pirmoji paraiška sistemoje.
            rodykle->tEileje = 0;
      else{
            //rodykle->setTAtejimo(rodykle2->t + rodykle->t);
            rodykle->tEileje = rodykle2->tEileje
                                                                        + rodykle2->t
                        + rodykle2->tAptarnavimo
                        - rodykle->t;
         if (rodykle->tEileje < 0)
            rodykle->tEileje = 0;
      }
      rodykle->tAptarnavimoPradzios = rodykle->t + rodykle->tEileje;
      // Paraiška palieka sistemą.
      // Sistema pereina į kitą būseną. Fiksuojamas eilės sumažėjimas.
      double mom = rodykle->t
                                                + rodykle->tEileje
                  + rodykle->tAptarnavimo;// Sumažėjimo momentas
      rodykle->IterptiMomenta(mom, -1);
      eilesN += 2;
      rodykle2 = rodykle;
   }

   //Form5->mm->Lines->Add("");
   //Form5->mm->Lines->Add("Sistema sumodeliuota.");
   //if (CheckBox1->Checked)
            //RodytiMasyva(N, NULL, Paraiskos, "Paraiškos ateina momentais:");
   /*
   if (CheckBox1->Checked){
            RodytiMasyva(N, NULL, Aptarnavimas, "Aptarnavimo trukmės:");
            RodytiMasyva(N, NULL, LaikasEileje, "Laikas eilėje:");
            RodytiMasyva(N, NULL, AptarnavimoPradzia, "Aptarnavimo pradžia:");
   }
   */
   if (CheckBox1->Checked){
            RodytiEilesPokycius(Eile, "Sistemos būsenų pokyčiai:");
   }
```

```
   // Sistemos būsenų pokyčių perskaičiavimas į busenų indeksus.
   maxBusena = 0;
   if (Eile)
             for (TEile * rod = Eile; rod->kita != NULL; rod = rod->kita){
                                       rod->kita->pokytis = rod->pokytis + rod->kita->pokytis;
         if (rod->kita->pokytis < 0)
              rod->kita->pokytis = 0;
         if (rod->kita->pokytis > maxBusena)
              maxBusena = rod->kita->pokytis;
      }
   //ShowMessage(maxBusena);
   if (Busenos) delete [] Busenos;
   Busenos = new double[maxBusena + 1];
   for (int i = 0; i <= maxBusena; i++)
             Busenos[i] = 0;
   // Sistemos būsenų indeksų perskaičiavimas į busenų indeksų trukmes.
   if (Eile)
             for (TEile * rod = Eile; rod->kita != NULL; rod = rod->kita){
                                       Busenos[rod->pokytis] += (rod->kita->t - rod->t);
         if (!rod->kita->kita)
              pabaiga = rod->kita->t;
      }
   //if (CheckBox1->Checked)
             //RodytiMasyva(maxBusena + 1, Busenos, NULL, "Sistemos būsenos atitinkamai truko:");

   if (CheckBox1->Checked)
             RodytiEilesPokycius(Eile, "Sistemos būsenos:");
   if (CheckBox1->Checked)
             RodytiMasyva(maxBusena + 1, NULL, Busenos, "Sistemos būsenų trukmės:");
}
//---------------------------------------------------------------------------
// Randamas vektoriaus vidurkis.
//---------------------------------------------------------------------------

double TForm1::Momentas(double * M, int n, int k)
{
             double suma = 0;
   for (int i = 0; i < n; i++)
             suma += pow(M[i], k);
   return suma / (double)n;
}
//---------------------------------------------------------------------------
// Randamos teorinės eilių sistemos charakteristikos.
//---------------------------------------------------------------------------

double * TForm1::TeoriniaiSistemosParametrai(double * parametrai)
{
             // Teorinės sistemos charakteristikos.

   // Eksponentinio aptarnavimo atvejis.
   double ro = liambdaP / liambdaA;
   double MParaiskos = 1 / liambdaP;
   double MALaikas = 1 / liambdaA;
   double MPSistemoje = ro / (1 - ro);
   double MPEileje = ro * ro / (1 - ro);
   double MTEileje = ro * ro / (liambdaP * (1 - ro));
   double MTSistemoje = ro / (liambdaP * (1 - ro));

   // Neeksponentinio aptarnavimmo atvejis.

   if (CompareStr(Sampler.DISTtype, "exp") != 0){

             double * Aptarnavimas = new double [N];
             TEile * eil = Eile;
             for (int i = 0; i < N; i++){
             Aptarnavimas[i] = eil->tAptarnavimo;
             eil = eil->kitaAteinanti;
             }

      double EA = Sampler.PradinisMomentas(1, Aptarnavimas, N);
      double EA2 = EA * EA;
      double v = (Sampler.PradinisMomentas(2, Aptarnavimas, N) - EA2) / EA2;
             liambdaA = 1. / EA;
             ro = liambdaP * EA;
             MParaiskos = 1 / liambdaP;
             MALaikas = EA;

      MPEileje = ro * ro * (1 + v) / (2 * (1 - ro));
             MPSistemoje = MPEileje + ro;
```

```
      MTEileje = ro * ro * (1 + v) / (2 * liambdaP * (1 - ro));
            MTSistemoje = MTEileje + EA;

      delete [] Aptarnavimas;
   }

   double * rez = NULL;
   if (parametrai){
            parametrai[0] = MParaiskos;
      parametrai[1] = MALaikas;
      parametrai[2] = MPSistemoje;
      parametrai[3] = MPEileje;
      parametrai[4] = MTEileje;
      parametrai[5] = MTSistemoje;
      parametrai[6] = ro;
   }
   else{
            rez = new double[7];
      rez[0] = MParaiskos;
      rez[1] = MALaikas;
      rez[2] = MPSistemoje;
      rez[3] = MPEileje;
      rez[4] = MTEileje;
      rez[5] = MTSistemoje;
      rez[6] = ro;
   }
   return rez;
}
//---------------------------------------------------------------------------
// Randamos empirinės eilių sistemos charakteristikos.
//---------------------------------------------------------------------------

double * TForm1::EmpiriniaiSistemosParametrai(double * parametrai)
{
   double sumaP = 0;
   double sumaA = 0;
   double sumaE = 0;
   double sumaE2 = 0;
   double sumaE3 = 0;
   double sumaEil = 0;
   double sumaSist = 0;

   for (TEile * rod = Eile; rod != NULL; rod = rod->kitaAteinanti){
            if (rod->kitaAteinanti != NULL)
                     sumaP += rod->kitaAteinanti->t - rod->t;
      sumaEil += rod->tEileje;
      sumaA += rod->tAptarnavimo;
      sumaSist += (rod->tAptarnavimo + rod->tEileje);
   }

   for (int i = 0; i <= maxBusena; i++)
            sumaE += Busenos[i];
   for (int i = 0; i <= maxBusena; i++){
            if (i > 1)
                     sumaE2 += (i - 1) * Busenos[i];
      sumaE3 += i * Busenos[i];
   }

   double MPSistemoje = sumaE3 / pabaiga;
   double MPEileje = sumaE2 / pabaiga;
   double MParaiskos = sumaP / double(N);
   double MALaikas = sumaA / double(N);
   double MTEileje = sumaEil / double(N);
   double MTSistemoje = sumaSist / double(N);
   double KUzimtas = 1 - Busenos[0] / pabaiga;

   double * rez = NULL;
   if (parametrai){
            parametrai[0] = MParaiskos;
      parametrai[1] = MALaikas;
      parametrai[2] = MPSistemoje;
      parametrai[3] = MPEileje;
      parametrai[4] = MTEileje;
      parametrai[5] = MTSistemoje;
      parametrai[6] = KUzimtas;
   }
   else{
            rez = new double[7];
      rez[0] = MParaiskos;
      rez[1] = MALaikas;
```

```
      rez[2] = MPSistemoje;
      rez[3] = MPEileje;
      rez[4] = MTEileje;
      rez[5] = MTSistemoje;
      rez[6] = KUzimtas;
   }
   return rez;
}
//---------------------------------------------------------------------------
// Lentelėje palyginamos eilių sistemos charakteristikos.
//---------------------------------------------------------------------------

void TForm1::PalygintiParametrus(double * teoriniai, double * empiriniai)
{
   double MParaiskos0      = teoriniai[0];
   double MALaikas0                    = teoriniai[1];
   double MPSistemoje0     = teoriniai[2];
   double MPEileje0                    = teoriniai[3];
   double MTEileje0                    = teoriniai[4];
   double MTSistemoje0     = teoriniai[5];
   double KUzimtas0                    = teoriniai[6];

   double MParaiskos          = empiriniai[0];
   double MALaikas                     = empiriniai[1];
   double MPSistemoje      = empiriniai[2];
   double MPEileje                     = empiriniai[3];
   double MTEileje                     = empiriniai[4];
   double MTSistemoje   = empiriniai[5];
   double KUzimtas                     = empiriniai[6];

   double epsMParaiskos      = fabs((empiriniai[0] - teoriniai[0]) / teoriniai[0]);
   double epsMALaikas        = fabs((empiriniai[1] - teoriniai[1]) / teoriniai[1]);
   double epsMPSistemoje  = fabs((empiriniai[2] - teoriniai[2]) / teoriniai[2]);
   double epsMPEileje        = fabs((empiriniai[3] - teoriniai[3]) / teoriniai[3]);
   double epsMTEileje        = fabs((empiriniai[4] - teoriniai[4]) / teoriniai[4]);
   double epsMTSistemoje  = fabs((empiriniai[5] - teoriniai[5]) / teoriniai[5]);
   double epsKUzimtas        = fabs((empiriniai[6] - teoriniai[6]) / teoriniai[6]);

   Form5->mm->Lines->Add("");
   Form5->mm->Lines->Add("+-----------------------------------------+----------+-------------+----
---------+");
   Form5->mm->Lines->Add("| Sistemos charakteristika                  |  Tiksli  | Sumodeliuota | S.
paklaida |");
   Form5->mm->Lines->Add("+-----------------------------------------+----------+-------------+----
---------+");

   Form5->mm->Lines->Add("| Vidutinis laikas tarp 2 ateinančių paraiškų |   "
            + FloatToStrF(MParaiskos0, ffFixed, 8, 4) + " |       "
      + FloatToStrF(MParaiskos, ffFixed, 8, 4) + " |      "
      + FloatToStrF(epsMParaiskos, ffFixed, 8, 4) + " |");
   Form5->mm->Lines->Add("| Vidutinis paraiškos aptarnavimo laikas      |   "
            + FloatToStrF(MALaikas0, ffFixed, 8, 4) + " |       "
      + FloatToStrF(MALaikas, ffFixed, 8, 4) + " |      "
      + FloatToStrF(epsMALaikas, ffFixed, 8, 4) + " |");

   Form5->mm->Lines->Add("+-----------------------------------------+----------+-------------+----
---------+");

   Form5->mm->Lines->Add("| Vidutinis paraiškų skaičius sistemoje       |   "
            + FloatToStrF(MPSistemoje0, ffFixed, 8, 4) + " |       "
      + FloatToStrF(MPSistemoje, ffFixed, 8, 4) + " |      "
      + FloatToStrF(epsMPSistemoje, ffFixed, 8, 4) + " |");
   Form5->mm->Lines->Add("| Vidutinis paraiškų skaičius eilėje          |   "
      + FloatToStrF(MPEileje0, ffFixed, 8, 4) + " |       "
      + FloatToStrF(MPEileje, ffFixed, 8, 4) + " |      "
      + FloatToStrF(epsMPEileje, ffFixed, 8, 4) + " |");

   Form5->mm->Lines->Add("+-----------------------------------------+----------+-------------+----
---------+");

   Form5->mm->Lines->Add("| Vidutinis paraiškos buvimo eilėje laikas    |   "
      + FloatToStrF(MTEileje0, ffFixed, 8, 4) + " |       "
      + FloatToStrF(MTEileje, ffFixed, 8, 4) + " |      "
      + FloatToStrF(epsMTEileje, ffFixed, 8, 4) + " |");
   Form5->mm->Lines->Add("| Vidutinis paraiškos buvimo sistemoje laikas |   "
      + FloatToStrF(MTSistemoje0, ffFixed, 8, 4) + " |       "
      + FloatToStrF(MTSistemoje, ffFixed, 8, 4) + " |      "
      + FloatToStrF(epsMTSistemoje, ffFixed, 8, 4) + " |");
```

```
   Form5->mm->Lines->Add("+-------------------------------------------+----------+-------------+----
---------+");

   Form5->mm->Lines->Add("| Tikimybė, kad kanalas užimtas              |   "
      + FloatToStrF(KUzimtas0, ffFixed, 8, 4) + " |       "
      + FloatToStrF(KUzimtas, ffFixed, 8, 4) + " |      "
      + FloatToStrF(epsKUzimtas, ffFixed, 8, 4) + " |");

   Form5->mm->Lines->Add("+-------------------------------------------+----------+-------------+----
---------+");
   Form5->Show();
   Form5->mm->Lines->Add("");
}
//---------------------------------------------------------------------------
// Laiko tarp ateinančių paraiškų tankio funkcija.
//---------------------------------------------------------------------------

void __fastcall TForm1::ppxPLOTClick(TObject *Sender)
{
            int n = StrToInt(Edit2->Text);
   double xn = StrToInt(Edit13->Text);
            double * x = new double [n];
   double * y = new double [n];
   for (int i = 0; i < n; i++){
            x[i] = 0 + xn * i / (double)n;
      y[i] = Sampler.pExp(x[i], liambdaP);
   }

   Form4->BraizytiTrajektorija(x, y, n, "FastLine", "x", "p(x)",

                                       clRed, "Įeinačių p. tankis");
   Form4->RodytiLegenda(true);
   Form4->Show();
}
//---------------------------------------------------------------------------
// Paraiškų aptarnavimo laiko tankio funkcija.
//---------------------------------------------------------------------------

void __fastcall TForm1::paxPLOTClick(TObject *Sender)
{
            int n = StrToInt(Edit2->Text);
   double xn = StrToInt(Edit13->Text);
            double * x = new double [n];
   double * y = new double [n];
   for (int i = 0; i < n; i++){
            x[i] = 0 + xn * i / (double)n;
      y[i] = Sampler.pTarget(x[i], liambdaA);
   }

   Form4->BraizytiTrajektorija(x, y, n, "FastLine", "x", "p(x)",

                                       clGreen, "Tikslinis tankis");
   Form4->RodytiLegenda(true);
   Form4->Show();
}
//---------------------------------------------------------------------------
// Alternatyvaus paraiškų aptarnavimo laiko tankio funkcija.
//---------------------------------------------------------------------------

void __fastcall TForm1::pqxPLOTClick(TObject *Sender)
{
            int n = StrToInt(Edit2->Text);
   double xn = StrToInt(Edit13->Text);
            double * x = new double [n];
   double * y = new double [n];
   for (int i = 0; i < n; i++){
            x[i] = 0 + xn * i / (double)n;
      y[i] = Sampler.pProposal(x[i], liambdaQ);
   }

   Form4->BraizytiTrajektorija(x, y, n, "FastLine", "x", "p(x)",

                                       clBlue, "Alternatyvus tankis");
   Form4->RodytiLegenda(true);
   Form4->Show();
}
//---------------------------------------------------------------------------
// Paraiškų aptarnavimo laiko pasiskirstymo funkcija.
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::FaxPLOTClick(TObject *Sender)
{
             int n = StrToInt(Edit2->Text);
   double xn = StrToInt(Edit13->Text);
             double * x = new double [n];
   double * y = new double [n];
   for (int i = 0; i < n; i++){
             x[i] = 0 + xn * i / (double)n;
      y[i] = Sampler.FTarget(x[i], liambdaA);
   }

   Form4->BraizytiTrajektorija(x, y, n, "FastLine", "x", "F(x)",

                                        clGreen, "Tikslinio tankio skirstinio f.");
   Form4->RodytiLegenda(true);
   Form4->Show();
}
//---------------------------------------------------------------------------
// Alternatyvaus paraiškų aptarnavimo laiko pasiskirstymo funkcija.
//---------------------------------------------------------------------------

void __fastcall TForm1::FqxPLOTClick(TObject *Sender)
{
             int n = StrToInt(Edit2->Text);
   double xn = StrToInt(Edit13->Text);
             double * x = new double [n];
   double * y = new double [n];
   for (int i = 0; i < n; i++){
             x[i] = 0 + xn * i / (double)n;
      y[i] = Sampler.FProposal(x[i], liambdaQ);
   }

   Form4->BraizytiTrajektorija(x, y, n, "FastLine", "x", "F(x)",

                            clBlue, "Alternatyvi pasiskirstymo f.");
   Form4->RodytiLegenda(true);
   Form4->Show();
}
//---------------------------------------------------------------------------
// Paraiškos laiko eilėje grafikas.
//---------------------------------------------------------------------------

void __fastcall TForm1::teilPLOTClick(TObject *Sender)
{
             double * Paraiskos = new double [N];
   double * LaikasEileje = new double [N];
   TEile * eil = Eile;
   for (int i = 0; i < N; i++){
             Paraiskos[i] = eil->t;
      LaikasEileje[i] = eil->tEileje;
      eil = eil->kitaAteinanti;
   }

             Form4->BraizytiTrajektorija(Paraiskos, LaikasEileje, N, "Bar",
                            "Paraiškos atėjimo momentas", "Laikas", clRed, "Laikas eilėje");
   Form4->RodytiLegenda(true);
   Form4->Show();

   delete [] Paraiskos;
   delete [] LaikasEileje;
}
//---------------------------------------------------------------------------
// Paraiškos aptarnavimo laiko grafikas.
//---------------------------------------------------------------------------

void __fastcall TForm1::taptPLOTClick(TObject *Sender)
{
             double * Paraiskos = new double [N];
   double * Aptarnavimas = new double [N];
   TEile * eil = Eile;
   for (int i = 0; i < N; i++){
             Paraiskos[i] = eil->t;
      Aptarnavimas[i] = eil->tAptarnavimo;
      eil = eil->kitaAteinanti;
   }

             Form4->BraizytiTrajektorija(Paraiskos, Aptarnavimas, N, "Bar",
                            "Paraiškos atėjimo momentas", "Laikas", clBlue, "Aptarnavimo laikas");
   Form4->RodytiLegenda(true);
   Form4->Show();
```

```
   delete [] Paraiskos;
   delete [] Aptarnavimas;
}
//---------------------------------------------------------------------------
// Programos darbo pabaiga.
//---------------------------------------------------------------------------

void __fastcall TForm1::Ieiti1Click(TObject *Sender)
{
              Close();
}
//---------------------------------------------------------------------------
// Grafiko saugojimas darbiniame kataloge (*.wmf formatu).
//---------------------------------------------------------------------------

void __fastcall TForm1::savePLOTClick(TObject *Sender)
{
              Form4->Chart1->SaveToMetafile(GetCurrentDir() + "\\Grafikas.wmf");
}
//---------------------------------------------------------------------------
// MCMC metodu sugeneruotų a.d. histograma.
//---------------------------------------------------------------------------

void __fastcall TForm1::pixPLOTClick(TObject *Sender)
{
              int n = StrToInt(Edit2->Text);
   double ** hist = M.Histograma(X, N, n);
   Form4->BraizytiTrajektorija(hist[0], hist[1], n, "Bar", "x", "¶(x)",

                                       clYellow, "MCMC histograma");
   Form4->RodytiLegenda(true);
   Form4->Show();
   M.TrintiDoubleMatrica(hist, 2);
}
//---------------------------------------------------------------------------
// MCMC metodu sugeneruotų a.d. kitimo grafikas.
//---------------------------------------------------------------------------

void __fastcall TForm1::pixPROCESSClick(TObject *Sender)
{
   double * I = new double [N];
   for (int i = 1; i <= N; i++)
              I[i] = i;
   Form4->BraizytiTrajektorija(I, X, N, "FastLine", "iteracija", "x", clGreen);
   Form4->RodytiLegenda(false);
   Form4->Show();
}
//---------------------------------------------------------------------------
// Trinami visi grafikai.
//---------------------------------------------------------------------------

void __fastcall TForm1::delPLOTSClick(TObject *Sender)
{
              Form4->TrintiGrafika();
}
//---------------------------------------------------------------------------
// Braižomas aproksimuojantis Erlango skirstinys.
//---------------------------------------------------------------------------

void __fastcall TForm1::ErlPLOTClick(TObject *Sender)
{
              int n = StrToInt(Edit2->Text);
   double xn = StrToInt(Edit13->Text);
              double * x = new double [n];
   double * y = new double [n];
   for (int i = 0; i < n; i++){
              x[i] = i * xn / (double)n;
      y[i] = Sampler.pErlang(x[i]);
   }
   Form4->BraizytiTrajektorija(x, y, n, "FastLine", "x", "Erl(x)",

                                       clGreen, "Erlango aproksimacija");
   Form4->RodytiLegenda(true);
   Form4->Show();
}
//---------------------------------------------------------------------------
// Aptarnavimo sistemos modeliavimas daud kartų.
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Button6Click(TObject *Sender)
{
              if (rezultatai){
              for (int i = 0; i < isViso; i++){
                            delete [] rezultatai[i];
              }
              delete [] rezultatai;
      rezultatai = NULL;
   }

              double nuo = StrToFloat(Edit15->Text);
   double kas = StrToFloat(Edit16->Text);
   double iki = StrToFloat(Edit17->Text);
   isViso = (int)((iki - nuo) / kas);
   while ((nuo + (isViso - 1) * kas) < iki) isViso++;
   rezultatai = new double * [isViso];

              Sampler.SetMemo(Form5->mm);
   Form5->Hide();

   // Kiek kartų modeliuoti su vienu parametrų rinkiniu.
   int kiek = StrToInt(Edit14->Text);

   for (int ii = 0; ii < isViso; ii++){
              //sb->SimpleText = "Modelling " + IntToStr(ii+1) + " of " + IntToStr(isViso) + ".";
              // Parametrų rinkiniai ir jų statistikos.
              // [0] Teoriniai.
              // [i] Modeliavimas.
              // ...
              // [kiek] Modeliavimas.
              // [kiek + 1] Vidurkis.
              // [kiek + 2] Standartas.
              // [kiek + 3] Epsilon.
              double ** parametrai = new double * [kiek + 4];
              for (int i = 0; i < kiek + 4; i++){
                            parametrai[i] = new double [7];
              for (int ii = 0; ii < 7; ii++)
                                          parametrai[i][ii] = 0;
              }
      // Parametrų skaitymas.
      SkaitytiParametrus2(nuo + ii*kas);
      if (RadioButton6->Checked)
              N = (int)(nuo + ii*kas);
      else
                            N = StrToInt(Edit1->Text);
              // Sistemos modeliavimai.
              for (int i = 0; i < kiek; i++){
              sb->SimpleText = "Modeliuojama. Parametras " + IntToStr(ii+1) + "/" + IntToStr(isViso) +
              ". Sistema " + IntToStr(i+1) + "/" + IntToStr(kiek) + ".";
                            ModeliuotiSistema();
              EmpiriniaiSistemosParametrai(parametrai[i + 1]);
              }
              // Teoriniai sistemos parametrai.
              TeoriniaiSistemosParametrai(parametrai[0]);
              // Parametrų statistikų radimas.
              if (kiek > 1)
                            for (int p = 0; p < 7; p++){
                                          // Vidurkis.
                                          for (int i = 1; i <= kiek; i++){
                                                        parametrai[kiek + 1][p] += parametrai[i][p];
                                          }
                            parametrai[kiek + 1][p] = parametrai[kiek + 1][p] / (double)kiek;
                            // Standartas.
                            for (int i = 1; i <= kiek; i++){
                                                        parametrai[kiek + 2][p] += pow(parametrai[i][p] -
parametrai[kiek + 1][p], 2);
                                          }
                            parametrai[kiek + 2][p] = sqrt(parametrai[kiek + 2][p] / (double)(kiek -
1));
                            // Epsilon.
                            parametrai[kiek + 3][p] = (parametrai[kiek + 1][p] - parametrai[0][p]) /
parametrai[0][p];
                            }
      rezultatai[ii] = parametrai[kiek + 3];
              if (CheckBox2->Checked)
              RodytiMatrica(parametrai, kiek + 4);
              for (int i = 0; i < kiek + 3; i++){  // Paskutinio netriname!!!
                            delete [] parametrai[i];
              }
              delete [] parametrai;
```

```
    }
    RodytiRezultatus(rezultatai, isViso);

    Form5->Show();
    Form5->mm->Lines->Add("");
    sb->SimpleText = "Atlikta.";
}
//---------------------------------------------------------------------------
// Aptarnavimo sistemos modeliavimo daug kartų rezultatų vaizdavimas.
//---------------------------------------------------------------------------

void TForm1::RodytiMatrica(double ** M, int eil)
{
   if (Form5->mm){
            AnsiString line = "Modeliavimo rezultatai:";
      Form5->mm->Lines->Add("");
      Form5->mm->Lines->Add(line);

      line = "+------------";
      for (int i = 0; i < 7; i++)
         line += "+----------";
      line += "+";
      AnsiString linija = line;

      Form5->mm->Lines->Add(linija);
      Form5->mm->Lines->Add("| Parametras |  Et(par) |  Et(apt) |  EPSist  |   EPEil  |   ETEil  |
ETSist  |  PUžimta |");
      Form5->mm->Lines->Add(linija);

      for (int i = 0; i < eil; i++){

         if (i == 0)
             line = "| Teoriškai  |";
         else
             if (i < eil - 3)
             line = "|            |";
         else
                        if (i == eil - 3)
                        line = "| Vidurkis   |";
             else
                        if (i == eil - 2)
                        line = "| Standartas |";
             else
                        if (i == eil - 1)
                                    line = "| Epsilon    |";
                        else
                                    line = "+------------|";

         for (int j = 0; j < 7; j++){
            AnsiString skc = FloatToStrF(M[i][j], ffFixed, 8, 4);
            int diff = 9 - skc.Length();
            while (diff){
              diff--;
                skc = " " + skc;
            }
            line += skc + " |";
         }
         if ((i == eil - 3) || (i == 1))
             Form5->mm->Lines->Add(linija);
         Form5->mm->Lines->Add(line);
      }

      Form5->mm->Lines->Add(linija);
   }
}
//---------------------------------------------------------------------------
// Aptarnavimo sistemos modeliavimo rezultatų vaizdavimas.
//---------------------------------------------------------------------------

void TForm1::RodytiRezultatus(double ** M, int eil)
{
   if (Form5->mm){
            AnsiString line = "Modeliavimo rezultatai (santykinės paklaidos):";
      Form5->mm->Lines->Add("");
      Form5->mm->Lines->Add(line);
      line = "+------------";
      for (int i = 0; i < 7; i++)
         line += "+----------";
      line += "+";
      AnsiString linija = line;
```

```
      Form5->mm->Lines->Add(linija);
      Form5->mm->Lines->Add("| Parametras |  Et(par) |  Et(apt) |  EPSist  |   EPEil  |   ETEil  |
ETSist  |  PUžimta |");
      Form5->mm->Lines->Add(linija);

      for (int i = 0; i < eil; i++){

         line = "+            |";
         for (int j = 0; j < 7; j++){
            AnsiString skc = FloatToStrF(M[i][j], ffFixed, 8, 4);
            int diff = 9 - skc.Length();
            while (diff){
              diff--;
                skc = " " + skc;
            }
            line += skc + " |";
         }
         Form5->mm->Lines->Add(line);
      }
      Form5->mm->Lines->Add(linija);
   }
}
//---------------------------------------------------------------------------
// Sistemos būsenų kitimo grafikas.
//---------------------------------------------------------------------------

void __fastcall TForm1::stagePLOTClick(TObject *Sender)
{
              double * Momentas = new double [eilesN];
   double * Busena = new double [eilesN];
   TEile * eil = Eile;
   for (int i = 0; i < eilesN; i++){
              Momentas[i] = eil->t;
      Busena[i] = eil->pokytis;
      //eil = eil->kitaAteinanti;
      eil = eil->kita;
   }

              Form4->BraizytiTrajektorija(Momentas, Busena, eilesN, "FastLine",
                          "Laiko momentas", "Sistemos būsena", clGreen,
         "Eilių sistemos būsenų kitimas");
   /*
   Form4->BraizytiTrajektorija(Momentas, Busena, eilesN, "Point",
                          "Laiko momentas", "Sistemos būsena", clGreen,
         "Eilių sistemos būsenų kitimas");
   */
   Form4->RodytiLegenda(true);
   Form4->Show();

   delete [] Momentas;
   delete [] Busena;
}
//---------------------------------------------------------------------------
// Skirtumai tarp teorinio ir alternatyvaus paraiškų aptarnavimo tankio f.
//---------------------------------------------------------------------------

void __fastcall TForm1::Epaxqax1Click(TObject *Sender)
{
              int n = StrToInt(Edit2->Text);
   double xn = StrToInt(Edit13->Text);
   double norma = 0;
              double * x = new double [n];
   double * y = new double [n];
   for (int i = 0; i < n; i++){
              x[i] = xn * i / (double)n;
      y[i] = fabs(Sampler.pProposal(x[i], liambdaQ)
                          - Sampler.pTarget(x[i], liambdaA));
      norma += y[i];
   }
   norma /= (double)n;

   Form4->BraizytiTrajektorija(x, y, n, "FastLine", "x", "Absoliuti paklaida",

                                       clBlue, "E(|pa(x) - qa(x)|)");
   Form4->RodytiLegenda(true);
   Form4->Show();

   Form5->mm->Lines->Add("E(|pa(x) - qa(x)|) = " + FloatToStr(norma));
}
//---------------------------------------------------------------------------
```

```
// Skirtumai tarp paraiškų aptarnavimo tankio funkcijos ir MCMC rezultato.
//---------------------------------------------------------------------------

void __fastcall TForm1::Epaxx1Click(TObject *Sender)
{
             int n = StrToInt(Edit2->Text);
             double * daznis = new double [n];
   for (int i = 0; i < n; i++) daznis[i] = 0;
   // hist [0; xn]
   int xn = StrToInt(Edit13->Text);
   //int dazniuSuma = 0;
   for (int i = 0; i < N; i++){
      double reiksme = X[i];
      for (int j = 1; j <= n; j++)
             if (reiksme < xn * j/(double)n){
             daznis[j-1] += 1;
            //dazniuSuma++;
            j = n + 1;
         }
   }
   for (int i = 0; i < n; i++)
                          daznis[i] = daznis[i] / (xn * N/(double)n);
   //ShowMessage("Dažnių suma: " + IntToStr(dazniuSuma));

   double norma = 0;
             double * x = new double [n];
   double * y = new double [n];
   for (int i = 0; i < n; i++){
             x[i] = xn * i / (double)n;
      y[i] = fabs(daznis[i] - Sampler.pTarget(x[i], liambdaA));
      norma += fabs(y[i]);
   }
   norma /= (double)n;

   Form4->BraizytiTrajektorija(x, y, n, "FastLine", "x", "Absoliuti paklaida",

                                       clGreen, "E(|pa(x) - ¶(x)|)");
   Form4->RodytiLegenda(true);
   Form4->Show();

   Form5->mm->Lines->Add("E(|pa(x) - ¶(x)|) = " + FloatToStr(norma));
}
//---------------------------------------------------------------------------
// Skirtumai tarp paraiškų aptarnavimo laiko tankio funkcijos ir jos
// aproksimacijos Erlango skirstiniu.
//---------------------------------------------------------------------------

void __fastcall TForm1::Epaxerlx1Click(TObject *Sender)
{
             int n = StrToInt(Edit2->Text);
   int xn = StrToInt(Edit13->Text);
   double norma = 0;
             double * x = new double [n];
   double * y = new double [n];
   for (int i = 0; i < n; i++){
             x[i] = xn * i / (double)n;
      y[i] = fabs(Sampler.pErlang(x[i])
                          - Sampler.pTarget(x[i], liambdaA));
      norma += y[i];
   }
   norma /= (double)n;

   Form4->BraizytiTrajektorija(x, y, n, "FastLine", "x", "Absoliuti paklaida",

                                       clRed, "E(|pa(x) - erl(x)|)");
   Form4->RodytiLegenda(true);
   Form4->Show();

   Form5->mm->Lines->Add("E(|pa(x) - erl(x)|) = " + FloatToStr(norma));
}
//---------------------------------------------------------------------------
// Informacija apie programą.
//---------------------------------------------------------------------------

void __fastcall TForm1::ApieClick(TObject *Sender)
{
             Form3->Show();
}
//---------------------------------------------------------------------------
// Programos darbo pradžia.
```

```
//---------------------------------------------------------------------------

void __fastcall TForm1::Form1Create(TObject *Sender)
{
             X = NULL;
   Eile = NULL;
   Busenos = NULL;
             rezultatai = NULL;
   MC.sb = this->sb;
   Sampler.sb = this->sb;
}
//---------------------------------------------------------------------------
// Vidutinio paraiškų skaičiaus eilėje paklaidos.
//---------------------------------------------------------------------------

void __fastcall TForm1::EPEilPaklaidosClick(TObject *Sender)
{
   double nuo = StrToFloat(Edit15->Text);
   double kas = StrToFloat(Edit16->Text);
   VidutiniuPaklaiduPriklausomybe(rezultatai, 3, nuo, kas, isViso);
}
//---------------------------------------------------------------------------
// Vidutinio paraiškų skaičiaus sistemoje paklaidos.
//---------------------------------------------------------------------------

void __fastcall TForm1::EPSistPaklaidosClick(TObject *Sender)
{
   double nuo = StrToFloat(Edit15->Text);
   double kas = StrToFloat(Edit16->Text);
   VidutiniuPaklaiduPriklausomybe(rezultatai, 2, nuo, kas, isViso);
}
//---------------------------------------------------------------------------
// Paraiškų laiko eilėje paklaidos.
//---------------------------------------------------------------------------

void __fastcall TForm1::ETEilPaklaidosClick(TObject *Sender)
{
   double nuo = StrToFloat(Edit15->Text);
   double kas = StrToFloat(Edit16->Text);
   VidutiniuPaklaiduPriklausomybe(rezultatai, 4, nuo, kas, isViso);
}
//---------------------------------------------------------------------------
// Paraiškų laiko sistemoje paklaidos.
//---------------------------------------------------------------------------

void __fastcall TForm1::ETSistPaklaidosClick(TObject *Sender)
{
   double nuo = StrToFloat(Edit15->Text);
   double kas = StrToFloat(Edit16->Text);
   VidutiniuPaklaiduPriklausomybe(rezultatai, 5, nuo, kas, isViso);
}
//---------------------------------------------------------------------------
// Vidutinių paklaidų priklausomybės tyrimas.
//---------------------------------------------------------------------------

void TForm1::VidutiniuPaklaiduPriklausomybe(double ** rezultatai, int id,


             double nuo, double kas, int n)
{
             double * x = new double [n];
   double * y = new double [n];
   double * z = new double [n];
   double * w = new double [n];
   for (int i = 0; i < n; i++){
             x[i] = nuo + i*kas;
      y[i] = fabs(rezultatai[i][id]);
   }
   // Slenkančio, kaip aproksimacijos, vidurkio radimas.
   for (int i = 0; i < n; i++){
             if (i == 0)
             w[i] = (y[i] + y[i+1]) / 2.;
      if (i > 0 && i < n - 1)
             w[i] = (y[i-1] + y[i] + y[i+1]) / 3.;
      if (i == n - 1)
             w[i] = (y[i-1] + y[i]) / 2.;
   }
   for (int i = 0; i < n; i++){
      z[i] = 0;
   }
```

```
   AnsiString label = "";
   AnsiString parametras = "Paraiškų atėjimo į sistemą intensyvumas";
   if (RadioButton6->Checked)
             parametras = "Atėjusių paraiškų skaičius";
   AnsiString label3 = "Slenkantis vidurkis";
   if (id == 2) label = "Paraiškų sk. sistemoje";
   if (id == 3) label = "Paraiškų sk. eilėje";
   if (id == 4) label = "Paraiškų laikas eilėje";
   if (id == 5) label = "Paraiškų laikas sistemoje";
   Form4->BraizytiTrajektorija(x, y, n, "Point",

                                       parametras, "| Santykinė paklaida |",

                                       clGreen, label);
   Form4->BraizytiTrajektorija(x, w, n, "FastLine",

                                       parametras, "| Santykinė paklaida |",

                                       clRed, label3);
   Form4->RodytiLegenda(true);
   Form4->Show();

   ofstream fr((label + ".txt").c_str());
   for (int i = 0; i < n; i++){
             fr << x[i] << "\t" << y[i] << "\t" << z[i] << "\t" << w[i];
      if (i < n - 1) fr << endl;
   }
   fr.close();

   delete [] x;
   delete [] y;
   delete [] z;
   delete [] w;
}
//---------------------------------------------------------------------------
// Akcijos kainų skaitymas.
//---------------------------------------------------------------------------

void __fastcall TForm1::Button1Click(TObject *Sender)
{
   M.mm = Form5->mm;
   MC.mm = Form5->mm;
   Form5->Visible = true;

   if (RadioButton7->Checked){
      M.OpenDialog1 = this->OpenDialog1;
      M.Skaityti("", true, true, Edit39, Edit40);
      MC.IvestiParametrus(M.Snul, M.ELnS, M.sigmaLnS);
   }
   else{
      double Snul = StrToFloat(Edit39->Text);
      double sigmaLnS = StrToFloat(Edit40->Text);
      //double ELnS = StrToFloat(Edit9->Text);
      M.IvestiParametrus(Snul, 0, sigmaLnS);
      MC.IvestiParametrus(Snul, 0, sigmaLnS);
   }
}
//---------------------------------------------------------------------------
// Pasirinktu metodu apskaičiuojama pasirinkimo sandorio kaina.
//---------------------------------------------------------------------------

void __fastcall TForm1::Button13Click(TObject *Sender)
{
   if (M.PV.kainosNuskaitytos){

      double i = StrToFloat(Edit18->Text);
      double XX = StrToFloat(Edit19->Text);
      int N2 = StrToInt(Edit20->Text);

      // Bus skaičiuojamas modeliavimo laikas
      clock_t laikas1, laikas2;
      laikas1 = clock();

      if (RadioButton24->Checked){
         M.RastiParametrus2(i, XX, N2, true);
         M.TiksliKaina();
      }
      if (RadioButton25->Checked){
         int epochos = StrToInt(Edit38->Text);
         int periodai = StrToInt(Edit30->Text);
```

```
         int trajektorijos = StrToInt(Edit34->Text);
         MC.IvestiParametrus(M.Snul, M.ELnS, M.sigmaLnS);
         MC.IvestiParametrus2(i, XX, N2);
         bool grafikas = CheckBox5->Checked;
         MC.MonteCarlo(epochos, trajektorijos, periodai, grafikas, true);
      }
      if (RadioButton9->Checked)
         if (M.PV.branduolinisTankis){
            if (M.PV.dalimisTolygusTankis){
               int epochos = StrToInt(Edit38->Text);
               int periodai = StrToInt(Edit30->Text);
               int trajektorijos = StrToInt(Edit34->Text);
               // Nauji parametrai
               MC.IvestiParametrus(M.Snul, M.ES, M.sigmaS);
               MC.IvestiParametrus2(i, XX, N2);
               bool grafikas = CheckBox6->Checked;
               // Vykdomas MCMC
               Sampler.kernelN = M.N4-1;
               Sampler.kernelH = M.kernelH;
               Sampler.kernelData = M.returns;
               //Sampler.kernelX = M.kernelX;
               //Sampler.kernelP = M.kernelP;
               Sampler.pLinearN = M.pLinearN;
               Sampler.pLinear = M.pLinear;
               Sampler.MCMCtype = "Independence piecewise";
               MCMCtype4->Checked = true;
                       int burn_in = StrToInt(Edit6->Text);
               double X0 = StrToFloat(Edit8->Text);
               Sampler.SetX0(X0);
                       Sampler.Burn_in(burn_in);
               N = trajektorijos * periodai;
               if (X) delete [] X; X = NULL;
               X = new double [N];
               if (Prop){
                  delete [] Prop;
                  Prop = NULL;
               }
               Prop = new double [N];
               // Atsitiktinių skaičių maišymui
               int * indexes = NULL;
               if (Shuffle->Checked){
                  indexes = new int [N];
                  for (int i = 0; i < N; i++){
                          indexes[i] = i;
                  }
                  //random_shuffle(indexes, indexes + N);
                  srand(unsigned(time(NULL)));
                  random_shuffle(indexes, indexes + N, p_myrandom);
                  for (int i = 0; i < N; i++){
                          X[indexes[i]] = Sampler.GetNumber();
                     Prop[indexes[i]] = Sampler.xi1;
                     //sb->SimpleText = "MCMC: " + IntToStr(N) + " \ " + IntToStr(i+1);
                  }
                  delete [] indexes;
                  indexes = NULL;
               }
               else{
                  for (int i = 0; i < N; i++){
                          X[i] = Sampler.GetNumber();
                     //sb->SimpleText = "MCMC: " + IntToStr(N) + " \ " + IntToStr(i+1);
                  }
               }
               /*
               // Sumaišytos sekos rašymas į failą (testavimas)
               ofstream log1("LOG-X-MCMC(shuffled).txt");
               for (int i = 0; i < N; i++){
                         log1 << X[i] << endl;
               }
               log1.close();
               */
               // MCMC pabaiga
               // MCMC rezultatai paduodami MC
               MC.MonteCarlo(epochos, trajektorijos, periodai, grafikas, true, X);
               //delete [] X;
            }
            else
               ShowMessage("Branduolinis tankis nesudarytas!");
         }
         else
            ShowMessage("Alternatyvus tankis nesudarytas!");
```

```
      laikas2 = clock();
      AnsiString laikas = AnsiString((laikas2 - laikas1) / CLK_TCK);
      Form5->mm->Lines->Add("_____________________________________________");
      Form5->mm->Lines->Add("Skaičiavimai užtruko: " + laikas + "s");
   }
   else
      ShowMessage("Ne visi duomenys įvesti!");
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button10Click(TObject *Sender)
{
   Form3->Visible = true;
   Form3->BringToFront();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::RadioButton18Click(TObject *Sender)
{
   M.tipas = "Call";
   MC.tipas = "Call";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::RadioButton19Click(TObject *Sender)
{
   M.tipas = "Put";
   MC.tipas = "Put";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
   M.SudarytiBranduoliniTanki(StrToFloat(Edit48->Text));
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Rodytilegend1Click(TObject *Sender)
{
   Form4->RodytiLegenda(true, true);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Rodytirezultatus1Click(TObject *Sender)
{
   Form5->Show();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Saugotirezultatus1Click(TObject *Sender)
{
   Form5->mm->Lines->SaveToFile("log.txt");
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Rodytigrafik1Click(TObject *Sender)
{
   Form4->Visible = true;
   Form4->BringToFront();
}
//---------------------------------------------------------------------------
// Akcijos grąžos branduolinio tankio įverčio vaizdavimas.
//---------------------------------------------------------------------------

void __fastcall TForm1::Button8Click(TObject *Sender)
{
   if (M.PV.branduolinisTankis){
      M.pLinearN = StrToInt(Edit47->Text) + 1;
      if (M.pLinear){
         delete [] M.pLinear[0];
         delete [] M.pLinear[1];
         delete [] M.pLinear[2];
         delete [] M.pLinear;
         M.pLinear = NULL;
      }
      M.pLinear = M.AlternatyvusSK(M.kernelX, M.kernelP, M.kernelN, M.pLinearN);
      M.PV.dalimisTolygusTankis = true;
      M.VaizduotiAlternatyvuTanki();
   }
```

```
   else
      ShowMessage("Branduolinis tankio įvertis nesudarytas!");
}
//---------------------------------------------------------------------------
// Akcijos grąžos logaritmo histograma.
//---------------------------------------------------------------------------

void __fastcall TForm1::Groslogaritmohistograma1Click(TObject *Sender)
{
   if (M.PV.istoriniaiDuomenys){
      int n = StrToInt(Edit2->Text);
      double ** hist = M.Histograma(M.returns, M.N4-1, n, true);
      //for (int i = 0; i < n - 1; i++)
         //hist[0][i] = (hist[0][i] + hist[0][i+1]) / 2.;
      Form4->BraizytiTrajektorija(hist[0], hist[1], n, "Bar", "x", "¶(x)",

                                        clYellow, "Realūs ln(grąža)");
      Form4->RodytiLegenda(true);
      Form4->Show();
      M.TrintiDoubleMatrica(hist, 2);
   }
   else
      ShowMessage("Iš failo nenuskaitytos aktyvo kainos!");
}
//---------------------------------------------------------------------------
// Akcijos grąžos branduolinio tankio vaizdavimas.
//---------------------------------------------------------------------------

void __fastcall TForm1::Groslogaritmobranduolinistankis1Click(
      TObject *Sender)
{
   if (M.PV.branduolinisTankis){
      M.VaizduotiBranduoliniTanki();
   }
   else
      ShowMessage("Branduolinis tankio įvertis nesudarytas!");
}
//---------------------------------------------------------------------------
// Akcijos istorinės kainos vaizdavimas.
//---------------------------------------------------------------------------

void __fastcall TForm1::Istorinkaina1Click(TObject *Sender)
{
   if (M.PV.istoriniaiDuomenys){
      //Form4->TrintiGrafika();
      Form4->BraizytiTrajektorija(M.indexes, M.prices, M.N4,
                                  "Line", "Data", "Akcijos kaina", clBlack,
                                  "Kainų grafikas");
      Form4->RodytiLegenda(true);
      Form4->Show();
   }
   else
      ShowMessage("Iš failo nenuskaitytos aktyvo kainos!");
}
//---------------------------------------------------------------------------
// Sumodeliuoto akcijos grąžos alternatyvus pasiskirstymo histograma.
//---------------------------------------------------------------------------

void __fastcall TForm1::Alternatyvausskhistograma1Click(TObject *Sender)
{
   if (Prop){
      int n = StrToInt(Edit2->Text);
      double ** hist = M.Histograma(Prop, N, n);
      Form4->BraizytiTrajektorija(hist[0], hist[1], n, "Bar", "x", "¶(x)",

                                        clYellow, "MCMC tankio f.");
      Form4->RodytiLegenda(true);
      Form4->Show();
      M.TrintiDoubleMatrica(hist, 2);
   }
   else
      ShowMessage("Alternatyvus pasiskirstymas nesumodeliuotas!");
}
//---------------------------------------------------------------------------
// Teorinio akcijos grąžos tankio, kurio parametrai lygūs istorinių akcijų
// kainų grąžų parametrams, vaizdavimas.
//---------------------------------------------------------------------------

void __fastcall TForm1::Teorinisgrostankis1Click(TObject *Sender)
{
```

```
   int n = 100;
   double sigma = M.sigmaLnS / sqrt(252.);
   double miu = M.ELnS / 252.;
   double x [100];
   double fx [100];
   double x1 = M.Minimum(M.returns, M.N4-1);
   double xn = M.Maximum(M.returns, M.N4-1);
   double dx = (xn - x1)/double(n-1);
   //ShowMessage(FloatToStr(x1) + " " + FloatToStr(dx) + " " + FloatToStr(xn));
   double PI = 2 * acos(0.0);

   for (int i = 0; i < n; i++){
      x[i] = x1 + i*dx;
      // Log-normal
      //exp(-pow(log(x) - miu, 2) / (2*sigma*sigma)) / (x*sigma*sqrt(2*PI));
      fx[i] = exp(-pow(log(x[i]+1) - miu, 2) / (2*sigma*sigma))
            / ((x[i]+1)*sigma*sqrt(2*PI));
   }

   Form4->BraizytiTrajektorija(x, fx, n, "Line", "x", "f(x)",

                                       clRed, "N(miu, sigma)");
   Form4->RodytiLegenda(true);
   Form4->Show();
}
//---------------------------------------------------------------------------
// Alternatyvaus akcijos grąžos pasiskirstymo vaizdavimas.
//---------------------------------------------------------------------------

void __fastcall TForm1::Alternatyviskirstiniofunkcija1Click(
      TObject *Sender)
{
   int histN = M.pLinearN;
   double * x = new double [histN];
   double * y = new double [histN];
   for (int i = 0; i < histN; i++){
      x[i] = M.pLinear[0][i];
      y[i] = M.pLinear[2][i];
   }
   //Form4->TrintiGrafika();
   Form4->BraizytiTrajektorija(x, y, histN, "Line", "x", "AltF(x)",

                                       clBlack, "Alternatyvus pasiskirstymas");
   Form4->RodytiLegenda(true);
   Form4->Show();
   delete [] x;
   delete [] y;
}
//---------------------------------------------------------------------------
// Optimalus branduolio plotis pagal Silverman.
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
   double h = 1.06 * M.sigmaS / (pow(M.N4, 1./5.) * sqrt(M.N4));
   Edit48->Text = FloatToStr(h);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::RadioButton5Click(TObject *Sender)
{
   Edit15->Text = FloatToStr(0.05);
   Edit16->Text = FloatToStr(0.02);
   Edit17->Text = FloatToStr(0.45);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::RadioButton6Click(TObject *Sender)
{
   Edit15->Text = 500;
   Edit16->Text = 2000;
   Edit17->Text = 50000;
}
//---------------------------------------------------------------------------
```

Eile.h

```
//---------------------------------------------------------------------------
#ifndef EileH
#define EileH
```

```
//---------------------------------------------------------------------------
class TEile
{
              //public:
              //int eilesIlgis;
      //int daznis;
              public:
              double t;  // Atėjimo/Išėjimo į/iš sistemą momentas.
      double tAptarnavimo;  // Aptarnavimo trukmė.
      double tAptarnavimoPradzios;  // Aptarnavimo pradžios momentas.
      double tEileje;  // Laokas eilėje.
      int pokytis;  // 1 - atėjo paraiška, -1 - išėjo paraiška.
      //int busena;  // Paraiškų skaičius sistemoje.
      TEile * kita;
      TEile * kitaAteinanti;  // Ne atminties valdymui, o tiesiog orientacijai.
      //TEile(int eilesIlgis, int daznis);
      TEile(double t, int pokytis);
      ~TEile();
      TEile * IterptiMomenta(double t, int pokytis);
};
//---------------------------------------------------------------------------
#endif
```

Eile.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Eile.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------
/*
TEile::TEile(int eilesIlgis, int daznis)
{
              this->eilesIlgis = eilesIlgis;
   this->daznis = daznis;
   this->t = 0;
   this->pokytis = 0;
   kita = NULL;
}
*/
//---------------------------------------------------------------------------

TEile::TEile(double t, int pokytis)
{
              //this->eilesIlgis = 0;
   //this->daznis = 0;

   tAptarnavimo = 0;
   tAptarnavimoPradzios = 0;
   tEileje = 0;

   this->t = t;
   this->pokytis = pokytis;
   //busena = pokytis;
   kita = NULL;
   kitaAteinanti = NULL;
}
//---------------------------------------------------------------------------

TEile::~TEile()
{
              //if (kita) delete kita;
}
//---------------------------------------------------------------------------
// Įterpiama nauja paraiška ir grąžinama rodyklė į ją.
//---------------------------------------------------------------------------

TEile * TEile::IterptiMomenta(double t, int pokytis)
{
   TEile * naujas = new TEile(t, pokytis);
              if (!kita){  // Ateityje eilės sumažėjimų nėra
      kita = naujas;
              if (pokytis == 1)
              kitaAteinanti = kita;
   }
   else{  // Ateityje eilės sumažėjimų yra
              TEile * rod = this;
              while (rod->kita->t < t){
```

```
            rod = rod->kita;
         if (!rod->kita)
            break;
      }
      naujas->kita = rod->kita;
      rod->kita = naujas;
      if (pokytis == 1){
            kitaAteinanti = naujas;
      }
   }
   return naujas;
}
//---------------------------------------------------------------------------
```

MCMC.h

```
//---------------------------------------------------------------------------
#ifndef MCMCH
#define MCMCH
//---------------------------------------------------------------------------
#include "math.h"
#include "Metodai.h"  // Bendrieji metodai
//---------------------------------------------------------------------------
const double PI = 2 * acos(0.0);
//---------------------------------------------------------------------------
class MCMC : public Metodai
{
            private:
            double lambdaP, lambdaA, lambdaQ;  // Exponential
            double LNTmiu, LNTsigma, LNPmiu, LNPsigma;  // Lognormal
      double ErlPk, ErlPliambda;  // Erlang
      double m1, m2, m3, u, v, miu1, miu2, p;  // Erlang
            private:
            double x0;
      double xi;
      TMemo * mm;
   public:
      TStatusBar * sb;
   public:
      //int kernelN;
      //double kernelH;
      //double * kernelX;
      //double * kernelP;
      int pLinearN;
      double ** pLinear;
      double xi1;
            public:
            MCMC();
      AnsiString DISTtype;
            AnsiString MCMCtype;
            void SetX0(double x0);
      void SetMemo(TMemo * mm);
      void SetLambdas(double lambdaP, double lambdaA, double lambdaQ);
      void SetLN(double LNTmiu, double LNTsigma, double LNPmiu, double LNPsigma);
      // Distributions
      double pExp(double x, double lambda);
            double invFExp(double x, double lambda);
      // Target distribution
      double FTarget(double x, double lambda);
      double pTarget(double x, double lambda);
      double invFTarget(double x, double lambda);
      // Proposal distribution
      double FProposal(double x, double lambda);
      double pProposal(double x, double lambda);
      double invFProposal(double x, double lambda);
      // Erlang distribution
      double pErlang(double x);
            double FErlang(double x);
      double invFErlang();
      // Kiti metodai
      double RandZ(double miu, double sigma);
      double Normal(double x);
      int fakt(int x);
            void Burn_in(int k);
      double PradinisMomentas(int k, double * X, int N);
      void InitErlang(int N);
      void MakeStep();
      double GetNumber();
};
//---------------------------------------------------------------------------
```

```
#endif
```

MCMC.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "MCMC.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------

MCMC::MCMC()
{
              mm = NULL;
              LNTmiu = 0.4;
              LNTsigma = 0.7;
              LNPmiu = 0.4;
              LNPsigma = 0.75;
}
//---------------------------------------------------------------------------

void MCMC::SetX0(double x0)
{
   this->x0 = x0;
}
//---------------------------------------------------------------------------

void MCMC::SetMemo(TMemo * mm)
{
              this->mm = mm;
}
//---------------------------------------------------------------------------

void MCMC::SetLambdas(double lambdaP, double lambdaA, double lambdaQ)
{
              this->lambdaP = lambdaP;
   this->lambdaA = lambdaA;
   this->lambdaQ = lambdaQ;
}
//---------------------------------------------------------------------------

void MCMC::SetLN(double LNTmiu, double LNTsigma, double LNPmiu, double LNPsigma)
{
              this->LNTmiu = LNTmiu;
   this->LNTsigma = LNTsigma;
   this->LNPmiu = LNPmiu;
   this->LNPsigma = LNPsigma;
}
//---------------------------------------------------------------------------
// Laiko tarp ateinančių paraiškų tankio funkcija.
//---------------------------------------------------------------------------

double MCMC::pExp(double x, double lambda)
{
              return lambda * exp(-lambda * x);
}
//---------------------------------------------------------------------------
// Atvirkštinė laiko tarp ateinančių paraiškų pasiskirstymo funkcija.
//---------------------------------------------------------------------------

double MCMC::invFExp(double x, double lambda)
{
              return -log(1 - x) / lambda;
}
//---------------------------------------------------------------------------

double MCMC::FTarget(double x, double lambda)
{
   // Gumbel
   //double laipsnis = exp(-lambda * x);
   //return exp(-laipsnis);

   if (DISTtype == "exp") // Exponential
              return exp(-lambda * x);

   if (DISTtype == "lognorm"){ // Lognormal
              if (x == 0) x = 0.0000000000000001;
                            return Normal((log(x) - LNTmiu) / LNTsigma);
   }
```

```
   return 0;
}
//---------------------------------------------------------------------------

double MCMC::pTarget(double x, double lambda)
{
            // Gumbel
   //double laipsnis = exp(-lambda * x);
   //return lambda * laipsnis * exp(-laipsnis);

            if (DISTtype == "exp") // Exponential
                        return lambda * exp(-lambda * x);

   if (DISTtype == "lognorm"){ // Lognormal
            if (x == 0) x = 0.0000000000000001;
            double trupmena = pow(log(x) - LNTmiu, 2) / (2 * pow(LNTsigma, 2));
            double eksponente = exp(-trupmena);
            return (1 / (x * LNTsigma * sqrt(2*PI))) * eksponente;
   }

   return 0;
}
//---------------------------------------------------------------------------

double MCMC::invFTarget(double x, double lambda)
{
   // Gumbel
   //return -log(-log(x)) / lambda;

   if (DISTtype == "exp") // Exponential
            return -log(1 - x) / lambda;

   if (DISTtype == "lognorm"){ // Lognormal
            //if (x == 0) x = 0.0000000000000001;
                        return exp(RandZ(LNTmiu, LNTsigma));
   }

   return 0;
}
//---------------------------------------------------------------------------

double MCMC::FProposal(double x, double lambda)
{
   // Gumbel
   //double laipsnis = exp(-lambda * x);
   //return exp(-laipsnis);

   if (DISTtype == "exp") // Exponential
            return exp(-lambda * x);

   if (DISTtype == "lognorm"){ // Lognormal
            if (x == 0) x = 0.0000000000000001;
                        return Normal((log(x) - LNPmiu) / LNPsigma);
   }

   return 0;
}
//---------------------------------------------------------------------------

double MCMC::pProposal(double x, double lambda)
{
   // Gumbel
   //double laipsnis = exp(-lambda * x);
   //return lambda * laipsnis * exp(-laipsnis);

   if (DISTtype == "exp") // Exponential
            return lambda * exp(-lambda * x);

   if (DISTtype == "lognorm"){ // Lognormal
            if (x == 0) x = 0.0000000000000001;
            double trupmena = pow(log(x) - LNPmiu, 2) / (2*pow(LNPsigma, 2));
            double eksponente = exp(-trupmena);
            return (1 / (x * LNPsigma * sqrt(2*PI))) * eksponente;
   }

   return 0;
}
//---------------------------------------------------------------------------
```

```
double MCMC::invFProposal(double x, double lambda)
{
              if (DISTtype == "exp") // Exponential
              return -log(1 - x) / lambda;
   if (DISTtype == "erl") // Erlang
              return invFErlang();
   if (DISTtype == "lognorm"){ // Lognormal
                          return exp(RandZ(LNPmiu, LNPsigma));
   }
   // Gumbel
   //return -log(-log(x)) / lambda;
   return 0;
}
//---------------------------------------------------------------------------

double MCMC::pErlang(double x)
{
              /*
              double expon1 = miu1 * exp(-miu1*x);
   double expon2 = -miu2*exp(-miu2*x);
   double expon3 = miu1*exp(-miu1*x);
   double expon4 = (expon2 + expon3) / (miu2-miu1);
   double big = p * miu1 * expon4;
   double reiksme = expon1 + big;
   if (reiksme < 0) reiksme = 0;
   */
   double reiskinys = pow(ErlPliambda, ErlPk) * pow(x, ErlPk - 1);
   double exponente = exp(-ErlPliambda * x);
   double reiksme = (reiskinys * exponente) / (fakt(ErlPk - 1));
              return reiksme;
}
//---------------------------------------------------------------------------

double MCMC::FErlang(double x)
{
              /*
              double expon = exp(-miu1*x);
   double big = p*miu1*(exp(-miu2*x)-exp(-miu1*x))/(miu2-miu1);
              return 1 - expon + big;
   */
   double reiksme = 0;
   for (int n = 0; n < ErlPk; n++){
              double reiskinys = pow(ErlPliambda * x, n);
              double exponente = exp(-ErlPliambda * x);
              reiksme += (reiskinys * exponente) / fakt(n);
   }
              return 1 - reiksme;
}
//---------------------------------------------------------------------------

double MCMC::invFErlang()
{
              /*
              double U1 = (double)rand() / ((double)(RAND_MAX + 1));
   double U2 = (double)rand() / ((double)(RAND_MAX + 1));
   double Y1 = invFExp(U1, miu1);
   double Y2 = invFExp(U2, miu2);
              return (Y1 + Y2) * p + Y1 * (1 - p);
   */
   double sum = 0;
   for (int n = 1; n <= ErlPk; n++){
              double U1 = (double)rand() / ((double)(RAND_MAX + 1));
              sum += invFExp(U1, ErlPliambda);
   }
   return sum;
}
//---------------------------------------------------------------------------
// Grąžna atsitiktinį skaičių ~ N(0, 1).
//---------------------------------------------------------------------------

double MCMC::RandZ(double miu, double sigma)
{
   double Z1 = (double)rand() / ((double)(RAND_MAX) + 1);
   double Z2 = (double)rand() / ((double)(RAND_MAX) + 1);
   double Z3 = (double)rand() / ((double)(RAND_MAX) + 1);
   double Z4 = (double)rand() / ((double)(RAND_MAX) + 1);
   double Z5 = (double)rand() / ((double)(RAND_MAX) + 1);
   double Z6 = (double)rand() / ((double)(RAND_MAX) + 1);

   double Z7 = (double)rand() / ((double)(RAND_MAX) + 1);
```

```
   double Z8 = (double)rand() / ((double)(RAND_MAX) + 1);
   double Z9 = (double)rand() / ((double)(RAND_MAX) + 1);
   double Z10 = (double)rand() / ((double)(RAND_MAX) + 1);
   double Z11 = (double)rand() / ((double)(RAND_MAX) + 1);
   double Z12 = (double)rand() / ((double)(RAND_MAX) + 1);

   double sum = Z1 + Z2 + Z3 + Z4 + Z5 + Z6 + Z7 + Z8 + Z9 + Z10 + Z11 + Z12;
   double rnd = (sum - 6);
   return rnd * sigma + miu;
}
//---------------------------------------------------------------------------
// Abromowitz and Stegun aproksimacija normaliajam skirstiniui (CDF).
//---------------------------------------------------------------------------

double MCMC::Normal(double x)
{
   double b1 =  0.319381530;
   double b2 = -0.356563782;
   double b3 =  1.781477937;
   double b4 = -1.821255978;
   double b5 =  1.330274429;
   double p  =  0.2316419;
   double c  =  0.39894228;

   if(x >= 0.0){
      double t = 1.0 / ( 1.0 + p * x );
      return (1.0 - c * exp( -x * x / 2.0 ) * t *
            ( t *( t * ( t * ( t * b5 + b4 ) + b3 ) + b2 ) + b1 ));
   }
   else{
      double t = 1.0 / ( 1.0 - p * x );
      return ( c * exp( -x * x / 2.0 ) * t *
            ( t *( t * ( t * ( t * b5 + b4 ) + b3 ) + b2 ) + b1 ));
   }
}
//---------------------------------------------------------------------------

int MCMC::fakt(int x)
{
              double f = 1;
   for (int i = x; i > 1; i--)
              f *= i;
   return f;
}
//---------------------------------------------------------------------------

void MCMC::Burn_in(int k)
{
   for (int i = 0; i <= k; i++){
              if (i == 0) xi = x0;
      if (i > 0) MakeStep();
   }
}
//---------------------------------------------------------------------------
// Pradinis skirstinio momentas.
//---------------------------------------------------------------------------

double MCMC::PradinisMomentas(int k, double * X, int N)
{
              double momentas = 0;
              if (DISTtype == "lognorm"){ // Lognormaliojo skirstinio momentas
              momentas = exp(k * LNTmiu + 0.5 * pow(k, 2) * pow(LNTsigma, 2));
      //double miu = log(LNTmiu);
      //double sigma= log(LNTsigma);
      //momentas = exp(k * miu + 0.5 * pow(k, 2) * pow(sigma, 2));
   }
              else if (DISTtype == "exp"){ // Eksponentinio skirstinio momentas
              momentas = fakt(k) / pow(lambdaA, k);
   }
   else{ // Nežinomo skirstinio momentas
              double sum = 0;
              for (int i = 0; i < N; i++){
              sum += pow(X[i], k);
              }
              momentas = sum / (double)N;
   }
   return momentas;
}
//---------------------------------------------------------------------------
// Aproksimuojama Erlango skirstiniu.
```

```
//---------------------------------------------------------------------------

void MCMC::InitErlang(int N)
{
              // Momentų radimas

   if (DISTtype == "lognorm" || DISTtype == "exp"){ // Pradiniai momentai
              m1 = PradinisMomentas(1, NULL, N);
              m2 = PradinisMomentas(2, NULL, N);
              m3 = PradinisMomentas(3, NULL, N);
   }
   else{
              double * X = new double [N];
              for (int i = 0; i < N; i++){
                            double unif = (double)rand() / ((double)(RAND_MAX + 1));
                            X[i] = invFTarget(unif, lambdaA);
              }
              m1 = PradinisMomentas(1, X, N);
              m2 = PradinisMomentas(2, X, N);
              m3 = PradinisMomentas(3, X, N);
      delete X;
              X = NULL;
   }

   // Informacija
   double variation = (m2 - m1*m1)/(m1*m1);
   if (mm){
      mm->Lines->Add("");
      mm->Lines->Add("Siūlomo skirstinio pradiniai momentai:");
              mm->Lines->Add("m1 = " + FloatToStr(m1));
              mm->Lines->Add("m2 = " + FloatToStr(m2));
              mm->Lines->Add("m3 = " + FloatToStr(m3));
              mm->Lines->Add("Variacija = "
                            + FloatToStr(variation)
              + " ("
              + FloatToStr(variation>=0.5)
              + ")");
   }
   //ShowMessage("Tikslinio tankio variacija: " + FloatToStr(variation));
   // Parametrų radimas
   /*
   u = (6 * m1 * m2 - 2 * m3) / (3 * pow(m2, 2) - 2 * m1 * m3);
   v = (12 * pow(m1, 2) - 6 * m2) / (3 * pow(m2, 2) - 2 * m1 * m3);
   miu1 = 0.5 * (u + sqrt(pow(u, 2) - 4 * v));
   miu2 = 0.5 * (u - sqrt(pow(u, 2) - 4 * v));
   p = (miu2 * (m1 * miu1 - 1)) / miu1;
   */
   double b = (m3 + m1 - 3*m2) / (m2 - m1);
   ErlPk = (int)b;
   ErlPliambda = pow((m2 - m1) / (b*b), 1 / b);
   if (mm){
              mm->Lines->Add("");
              mm->Lines->Add("Erlango skirstinio parametrai:");
      /*
              mm->Lines->Add(" u =\t" + FloatToStr(u));
      mm->Lines->Add(" v =\t" + FloatToStr(v));
      mm->Lines->Add(" miu1 =\t" + FloatToStr(miu1));
      mm->Lines->Add(" miu2 =\t" + FloatToStr(miu2));
      mm->Lines->Add(" p =\t" + FloatToStr(p));
      */
      mm->Lines->Add(" Erlang k =\t" + FloatToStr(ErlPk));
      mm->Lines->Add(" Erlang liambda =\t" + FloatToStr(ErlPliambda));
   }
   //ShowMessage(u);
   //ShowMessage(FloatToStr(m1) +" "+ FloatToStr(m2) +" "+ FloatToStr(m3));
   //ShowMessage("Erl(" + FloatToStr(ErlPk) + ", " + FloatToStr(ErlPliambda) + ")");
}
//---------------------------------------------------------------------------

void MCMC::MakeStep()
{
              //double xi1;
   //double U1 = (double)rand() / ((double)(RAND_MAX + 1));
   double U1 = doubleParkMiller();
   //ShowMessage(U1);
   // proposal
   if (MCMCtype == "Independence sampler")  // Independence sampler
              xi1 = invFProposal(U1, lambdaQ);
   else if (MCMCtype == "Independence piecewise")
              xi1 = ATInvReiksme(pLinear[0], pLinear[2], pLinearN, U1);
```

```
   else  // Erlang approximation
      xi1 = invFErlang();

   // Generavimo vaizdavimas
   /*
   double * masX = new double[3];
   double * masY = new double[3];
   masX[0] = -0.2;
   masX[1] = xi1;
   masX[2] = xi1;
   masY[0] = U1;
   masY[1] = U1;
   masY[2] = 0;
   Form4->BraizytiTrajektorija(masX, masY, 3, "Line", "x", "prop(x)",

                                        clGray, "");
   delete [] masX; masX = NULL;
   delete [] masY; masY = NULL;
   */

   // acceptance
   double skaitiklis = 0;
   double vardiklis = 0;
   double alfa = 0;
   //ShowMessage(pTarget(xi1, lambdaA));
   //ShowMessage(pProposal(xi, lambdaQ));
   //ShowMessage(pProposal(xi1,lambdaQ));
   //ShowMessage(pTarget(xi, lambdaA));
   //ShowMessage(kernelH);
   if (MCMCtype == "Erlang approximation"){
            skaitiklis = pTarget(xi1, lambdaA) * pErlang(xi);
      vardiklis = pTarget(xi, lambdaA) * pErlang(xi1);
   }
   else if (MCMCtype == "Independence piecewise"){
      double s1 = fbx(kernelData, kernelN, xi1, kernelH);
      double s2 = ATReiksme(pLinear[0], pLinear[1], pLinearN, xi);
      double v1 = fbx(kernelData, kernelN, xi, kernelH);
      double v2 = ATReiksme(pLinear[0], pLinear[1], pLinearN, xi1);
            skaitiklis = s1 * s2;
      vardiklis = v1 * v2;
      /*
      ShowMessage("fbx(xi1) = " + FloatToStr(s1)
            + " ATReiksme(xi) = " + FloatToStr(s2)
            + " fbx(xi) = " + FloatToStr(v1)
            + " ATReiksme(xi1) = " + FloatToStr(v2));
      */
   }
   else{
            skaitiklis = (pTarget(xi1, lambdaA) * pProposal(xi, lambdaQ));
      vardiklis = pTarget(xi, lambdaA) * pProposal(xi1,lambdaQ);
   }
   if (vardiklis == 0) vardiklis = 0.0000000000000001;
   alfa = skaitiklis / vardiklis;
   if (alfa > 1) alfa = 1;
   //double U2 = (double)rand() / ((double)(RAND_MAX + 1));
   double U2 = doubleParkMiller();
   if (U2 <= alfa) xi = xi1;
   //else MakeStep();  // Be neįvykusio šuoliuko.
   /*
   ShowMessage("x(i): " + FloatToStr(xi)
            + " Proposal: " + FloatToStr(xi1)
            + " alfa: " + FloatToStr(alfa)
            + " skaitiklis: " + FloatToStr(skaitiklis)
            + " vardiklis: " + FloatToStr(vardiklis));
   */
   /*
   if (mm){
      mm->Lines->Add("-------------------------------------");
      mm->Lines->Add("x(i): " + FloatToStr(xi));
      mm->Lines->Add("Proposal: " + FloatToStr(xi1));
      mm->Lines->Add("alfa: " + FloatToStr(alfa));
   }
   */
}
//---------------------------------------------------------------------------

double MCMC::GetNumber()
{
            MakeStep();
            return xi;
```

```
}
//---------------------------------------------------------------------------
```

MCarlo.h

```
//---------------------------------------------------------------------------
#ifndef MCarloH
#define MCarloH
//---------------------------------------------------------------------------
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include "Math.h"
#include "Metodai.h"  // Bendrieji metodai
#include "Grafikas.h"
//---------------------------------------------------------------------------
// Klasė Monte Karlo metodui valdyti.
//---------------------------------------------------------------------------
class MCarlo : public Metodai
{
   private:
      int Nper;  // Sandorio trukmė (mėn)
      int Nrea;  // Realizacijų skaičius
      int Nepochoje;  // Realizacijų 1 epochoje skaičius
      int Nepo;  // Epochų skaičius
      int m, n, mn;  // Matricos ir masyvo išmatavimai
      double Snul, i, X;
      double miu, sigma;
      double ** Vertes;
      double * Kainos;
   public:
      TMemo * mm;
      TStatusBar * sb;
      AnsiString tipas;  // Sandorio tipas (Call, Put)
   public:
      MCarlo();
      ~MCarlo();
      void TrintiAtminti();
      void Epocha(double Snul, int periodai, int epocha, double * XX = NULL);
      double MonteCarlo(int Nepo, int Nepochoje, int periodai,
                        bool grafikas, bool log, double * XX = NULL);
      double Kaina(int periodai);
      void IvestiParametrus(double Snul, double ELnS, double sigmaLnS);
      void IvestiParametrus2(double i, double X, int Nper);
};
//---------------------------------------------------------------------------
#endif
```

MCarlo.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "MCarlo.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------
// Klasės konstruktorius. Pradinių reikšmių priskyrimas kintamiesiems.
//---------------------------------------------------------------------------

MCarlo::MCarlo()
{
   Metodai();

   Nper = Nrea = Nepochoje = Nepo = 0;
   m = n = mn = 0;
   miu = sigma = 0;
   Snul = 0;
   Vertes = NULL;
   Kainos = NULL;
   tipas = "Call";
}
//---------------------------------------------------------------------------
// Klasės destruktorius.
//---------------------------------------------------------------------------

MCarlo::~MCarlo()
{
   TrintiAtminti();
}
//---------------------------------------------------------------------------
```

```
// Atlaisvinama dinaminė atmintis.
//---------------------------------------------------------------------------

void MCarlo::TrintiAtminti()
{
   TrintiDoubleMatrica(Vertes, m);
   Vertes = NULL;
   m = 0;
   if (Kainos){
      delete [] Kainos;
      Kainos = NULL;
      mn = 0;
   }
}
//---------------------------------------------------------------------------
//  Snul - pradinė kaina.
//  epocha - epochos indeksas: 0; 1; 2; ...
//---------------------------------------------------------------------------

void MCarlo::Epocha(double Snul, int periodai, int epocha, double * XX)
{
   double t = Nper / (252. * (double)periodai);
   double palukanos = log(1 + i);  // delta
   double miu = palukanos - sigma * sigma / 2.;
   double miut = miu * t;
   double sigmasqrtt = sigma * sqrt(t);
   double laipsnis = 0;
   double * XX2 = NULL;

   if (XX){
      XX2 = new double[(n-1)*m];
      double miu = VektoriausVidurkis(XX, (n-1)*m);
      double sigma = VektoriausStandartas(XX, (n-1)*m, miu);
      //ShowMessage("miu(XX) = " + FloatToStr(miu) + "\nsigma(XX) = " + FloatToStr(sigma));
      // START: Normalizavimas
      for (int i = 0; i < (n-1)*m; i++)
         XX2[i] = (XX[i] - miu)/ sigma;  // Normalizuojamos dieninės
      // END: Normalizavimas
      //miut = miu * t;
      //sigmasqrtt = sigma * sqrt(t);
      miut = this->miu * t;
      sigmasqrtt = this->sigma * sqrt(t);
   }

   for (int i = 0; i < m; i++){
      for (int j = 1; j < n; j++){
         if (XX){
         // (n-2), nes S0 nereikia atstiktinio skaičiaus
            laipsnis = miut + sigmasqrtt * XX2[(n-2) * i + j];  // Normalizuojama prieš trajektorijas
            //laipsnis = XX[(n-1) * i + j - 1];  // Nenormalizuojama sudarant branduolini
            //ShowMessage("XX[" + FloatToStr((n-1) * i + j - 1) + "] = " + FloatToStr(XX[(n-1) * i + j
- 1]));
            Vertes[i][j] = Vertes[i][j - 1] * (1 + laipsnis);
            //Vertes[i][j] = Vertes[i][j - 1] * laipsnis;
         }
         else{
            laipsnis = miut + sigmasqrtt * RandZ();
            Vertes[i][j] = Vertes[i][j - 1] * exp(laipsnis);
         }
         //AnsiString info = AnsiString(i) + " -> " + AnsiString(j);
         //sb->SimpleText = info;
      }
      Kainos[m * epocha + i] = Vertes[i][n - 1];
   }
   if (XX2){
      delete [] XX2;
      XX2 = NULL;
   }
}
//---------------------------------------------------------------------------
// Metodas valdo Monte Karlo modeliavimą.
//---------------------------------------------------------------------------

double MCarlo::MonteCarlo(int Nepo, int Nepochoje, int periodai,
                          bool grafikas, bool log, double * XX)
{
   if (Kainos && Vertes)
      TrintiAtminti();

   this->Nepochoje = Nepochoje;
```

```
   this->Nepo = Nepo;
   Nrea = Nepochoje * Nepo;

   m = Nepochoje;
   n = periodai + 1;  // Nepamirštam Snul-nio
   mn = Nrea;

   double * indeksai = new double[n];
   for (int i = 0; i < n; i++){
      indeksai[i] = i;
   }

   Kainos = new double [mn];
   Vertes = DoubleMatrica(m, n);

   for (int j = 0; j < m; j++){
      Vertes[j][0] = Snul;
   }

   if (grafikas){
      Form4->TrintiGrafika();
   }

   //randomize();
   for (int i = 0; i < Nepo; i++){
      Epocha(Snul, periodai, i, XX);
      if (grafikas && i == 0){
         for (int j = 0; j < m && j < 500; j++){
            Form4->BraizytiTrajektorija(indeksai, Vertes[j], n, "Line", "Periodas", "Kaina", clBlack);
         }
      }
   }

   if (grafikas){
      Form4->RodytiLegenda(false);
      Form4->Visible = true;
      Form4->BringToFront();
   }

   double C = Kaina(periodai);

   if (log){
      mm->Lines->Add("");
      mm->Lines->Add("MonteCarlo modeliavimas");
      mm->Lines->Add("");
      mm->Lines->Add("Sandorio kaina: " + FormatFloat("0.0000", C));
   }
   delete [] indeksai;
   return C;
}
//---------------------------------------------------------------------------
// Masyve Kainos() yra surašytos kainos STi, belieka rasti C.
//---------------------------------------------------------------------------

double MCarlo::Kaina(int periodai)
{
   double suma = 0;
   double vidurkis = 0;
   double kaina = 0;
   double t = Nper / 252.;
   //double t = Nper / (252. * (double)periodai);
   double delta = log(1 + i);

   for (int i = 0; i < mn; i++){
      suma += Ismoka(Kainos[i], X, tipas);
   }

   vidurkis = suma / (double)mn;
   kaina = vidurkis * exp(-delta * t);
   return kaina;
}
//---------------------------------------------------------------------------
// Įvedami modelio parametrai.
//---------------------------------------------------------------------------

void MCarlo::IvestiParametrus(double Snul, double ELnS, double sigmaLnS)
{
   this->Snul = Snul;
   this->miu = ELnS;
   this->sigma = sigmaLnS;
```

```
}
//---------------------------------------------------------------------------
// Įvedami modelio parametrai.
//---------------------------------------------------------------------------

void MCarlo::IvestiParametrus2(double i, double X, int Nper)
{
   this->Nper = Nper;  // Sandorio trukmė (d.)
   this->i = i;        // Paprastoji nerizikingoji palūkanų norma
   this->X = X;        // Sandorio Įvykdymo kaina
}
//---------------------------------------------------------------------------
```

Modeliavimas.h

```
//---------------------------------------------------------------------------
#ifndef ModeliavimasH
#define ModeliavimasH
//---------------------------------------------------------------------------
#include <time.h>
#include <fstream>
#include <iomanip>
using namespace std;
//---------------------------------------------------------------------------
#include "Forma5.h"  // Rezultatų langas
#include "Metodai.h"  // Bendrieji metodai
#include "Math.h"
//---------------------------------------------------------------------------
// Apsaugai, jei bandoma vykdyti programą nenuskaičius duomenų.
//---------------------------------------------------------------------------
struct PadeciuVektorius
{
   bool atmintisIsskirta;
   bool kainosNuskaitytos;
   bool parametraiIvesti;
   bool istoriniaiDuomenys;
   bool branduolinisTankis;
   bool dalimisTolygusTankis;
};
//---------------------------------------------------------------------------
// Kalsė binominiam ir Black-Scholes metodams valdyti.
//---------------------------------------------------------------------------
class Modelis : public Metodai
{
   private:
      AnsiString filename;  // Duomenų failo vardas
      AnsiString assetname;  // Aktyvo pavadinimas
      double * CC;  // Sandorio kainų masyvas
      double * NN;  // Gardelės periodų masyvas
      double E, sigma;  // Vienos dienos kainoms
      //
      double i, deltaT, R, q, u, d, X, C;
      int N1;  // Periodų skaičius + 1 (gardelei)
      int N2;  // Sandorio trukmė (d.)
   public:
      double Snul;  // Pradinė kaina
      double ES, sigmaS;  // Metinėms grąžoms
      double ELnS, sigmaLnS;  // Metiniai
      AnsiString tipas;  // Sandorio tipas (Call, Put)
      double * indexes;  // Indeksų masyvas
      AnsiString * dates;  // Datų iš failo masyvas
      double * prices;  // Kainų iš failo masyvas
      double * returns;  // Kainų grąžų masyvas
      double * returnsL;  // Kainų grąžų logaritmų masyvas
      //double * Nreturns;  // Normalizuotų kainų grąžų masyvas
      int N4;  // Duomenų failo eilučių skaičius
   public:
      int pLinearN;
      double ** pLinear;
   public:
      bool dots;
      TMemo * mm;
      PadeciuVektorius PV;
   private:
      void IsskirtiAtminti(int);
      void TrintiAtminti();
   public:
      Modelis();
      ~Modelis();
      void Skaityti(AnsiString fn, bool opendialog, bool logg,
```

```
                        TEdit * ee1, TEdit * ee2);
      void RastiMiuSigma(bool logg, TEdit * ee1, TEdit * ee2);
      void RastiParametrus2(double, double, int, bool);
      void IvestiParametrus(double, /* double, double, int, int, */ double, double);
      void TiksliKaina();
      double TiksliKaina(double, int);
      void SudarytiBranduoliniTanki(double kernelH);
      void VaizduotiBranduoliniTanki();
      void VaizduotiAlternatyvuTanki();
};
//---------------------------------------------------------------------------
#endif
```

Modeliavimas.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Modeliavimas.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------
// Klasės konstruktorius. Pradinių reikšmių priskyrimas kintamiesiems.
//---------------------------------------------------------------------------

Modelis::Modelis()
{
   Metodai();

   PV.atmintisIsskirta =  false;
   PV.kainosNuskaitytos = false;
   PV.parametraiIvesti =  false;
   PV.istoriniaiDuomenys = false;
   PV.branduolinisTankis = false;
   PV.dalimisTolygusTankis = false;

   N1 = N2 = N4 = 0;
   dots = false;
   filename = "";
   assetname = "";
   tipas = "Call";

   dates = NULL;
   prices = NULL;
   returns = NULL;
   returnsL = NULL;
   indexes = NULL;
   mm = NULL;
   OpenDialog1 = NULL;

   kernelX = NULL;
   kernelP = NULL;
   pLinear = NULL;
}
//---------------------------------------------------------------------------
// Klasės destruktorius.
//---------------------------------------------------------------------------

Modelis::~Modelis()
{
   TrintiAtminti();
}
//---------------------------------------------------------------------------
// Sukuriami dinaminiai masyvai išfailo nuskaitomoms datoms ir kainoms.
//---------------------------------------------------------------------------

void Modelis::IsskirtiAtminti(int kiek)
{
   dates = new AnsiString [kiek];
   prices = new double [kiek];
   returns = new double [kiek-1];
   returnsL = new double [kiek-1];
   indexes = new double [kiek];

   PV.atmintisIsskirta = true;
}
//---------------------------------------------------------------------------
// Datų, kainų ir jų indeksų masyvų užimamos atminties atlaisvinimas.
//---------------------------------------------------------------------------

void Modelis::TrintiAtminti()
```

```
{
   if (dates){
      delete [] dates;
      dates = NULL;
      PV.istoriniaiDuomenys = false;
   }
   if (prices){
      delete [] prices;
      prices = NULL;
      PV.istoriniaiDuomenys = false;
   }
   if (returns){
      delete [] returns;
      returns = NULL;
      PV.istoriniaiDuomenys = false;
   }
   if (returnsL){
      delete [] returnsL;
      returnsL = NULL;
      PV.istoriniaiDuomenys = false;
   }
   if (indexes){
      delete [] indexes;
      indexes = NULL;
   }
   if (kernelX){
      delete [] kernelX;
      kernelX = NULL;
   }
   if (kernelP){
      delete [] kernelP;
      kernelP = NULL;
   }
   if (pLinear){
      delete [] pLinear[0];
      delete [] pLinear[1];
      delete [] pLinear[2];
      delete [] pLinear;
      pLinear = NULL;
   }
   PV.atmintisIsskirta = false;
}
//---------------------------------------------------------------------------
// Iš failo skaitomos datos ir kainos. Failo vardas imamas kaip aktyvo vardas.
//---------------------------------------------------------------------------

void Modelis::Skaityti(AnsiString fn, bool opendialog, bool logg,
                       TEdit * ee1, TEdit * ee2)
{
   if (opendialog)
      filename = DFN();
   else
      filename = fn;

   assetname = FailoVardas(filename);
   if (logg){
      AnsiString line = "Pasirinktas aktyvas: " + assetname;
      mm->Lines->Add(line);
      mm->Lines->Add("");
   }

   if (filename != ""){

      int i = 0;
      char date [10];
      ifstream fd(filename.c_str());
      fd >> N4;
      fd.ignore();
      if (PV.atmintisIsskirta)
         TrintiAtminti();
      IsskirtiAtminti(N4);

      while (i < N4 && fd.good()){
         fd >> date >> prices[i];
         dates[i++] = AnsiString(date);
         fd.ignore();
      }
      fd.close();
      PV.kainosNuskaitytos = true;
      PV.istoriniaiDuomenys = true;
```

```
      if (logg){
         AnsiString line = "";
         for (int i = 0; i < N4; i++){
            if (i <= 5 || i > N4 - 4){
               line = AnsiString(i) + ". " + AnsiString(dates[i]) + " ~~~ " + AnsiString(prices[i]);
               mm->Lines->Add(line);
               line = "";
            }
            else
               if (i > 10 && i <= 13)
                  mm->Lines->Add("...");
            indexes[i] = -N4+i+1;
         }
      }
      RastiMiuSigma(logg, ee1, ee2);
      PV.branduolinisTankis = false;
      PV.dalimisTolygusTankis = false;
   }
}
//---------------------------------------------------------------------------
// Iš failo nuskaitytų duomenų kainų grąžų logaritmų vidurkio ir
// standarto radimas.
//---------------------------------------------------------------------------

void Modelis::RastiMiuSigma(bool logg, TEdit * ee1, TEdit * ee2)
{
   double sum = 0;
   double sum1 = 0;
   double sum2 = 0;
   double sum3 = 0;
   double sum4 = 0;
   double sum5 = 0;

   for (int i = 0; i < N4; i++){
      sum += prices[i];
      if (i > 0){
         returnsL[i-1] = log(prices[i] / prices[i - 1]);
         returns[i-1] = prices[i] / prices[i - 1] - 1;
         sum1 += returnsL[i-1];
         sum4 += returns[i-1];
      }
   }
   // Vidurkiai
   E = sum / (double)N4;
   ES = sum4 / (double)(N4 - 1);
   ELnS = sum1 / (double)(N4 - 1);

   for (int i = 0; i < N4; i++){
      sum2 += pow(prices[i] - E, 2);
      if (i < N4-1){
         sum3 += pow(returnsL[i] - ELnS, 2);
         sum5 += pow(returns[i] - ES, 2);
      }
   }
   ES *= 252;
   ELnS *= 252;
   // Standartiniai nuokrypiai (iš karto metiniai)
   sigma = sqrt(252 * sum2 / (N4 - 1));
   sigmaS = sqrt(252 * sum5 / (double)(N4 - 2));
   sigmaLnS = sqrt(252 * sum3 / (double)(N4 - 2));
   /*
   // Grąžų logaritmų normalizavimas
   for (int i = 0; i < N4-1; i++){
      returnsL[i] = (returnsL[i] - ELnS) / sigmaLnS;
   }
   */

   Snul = prices[N4 - 1];
   ee1->Text = FloatToStr(Snul);
   ee2->Text = FloatToStr(sigmaLnS);

   if (logg){
      AnsiString eilute = "";
      mm->Lines->Add("");
      mm->Lines->Add("E(dienos kainai)= " + AnsiString(E));
      mm->Lines->Add("sigma(dienos kainai) = " + AnsiString(sigma));
      mm->Lines->Add("");
      mm->Lines->Add("E(LnS) = " + AnsiString(ELnS));
      mm->Lines->Add("sigma(LnS) = " + AnsiString(sigmaLnS));
```

```
      mm->Lines->Add("");
      mm->Lines->Add("E(metinei gražai) = " + AnsiString(ES));
      mm->Lines->Add("sigma(metinei gražai) = " + AnsiString(sigmaS));
   }
}
//---------------------------------------------------------------------------
// Įvedami i, X ir sandorio trukmė.
//---------------------------------------------------------------------------

void Modelis::RastiParametrus2(double i, double X, int N2, bool logg)
{
   this->N2 = N2;  // Sandorio trukmė (d.)
   this->i = i;
   this->X = X;

   if (logg){
      mm->Lines->Add("");
      mm->Lines->Add("Palūkanų norma: " + FloatToStr(this->i));
      mm->Lines->Add("Įvykdymo kaina: " + FloatToStr(this->X));
      mm->Lines->Add("Sandorio trukmė : " + FloatToStr(N2) + " dienos(-ų)");
   }

   PV.parametraiIvesti = true;

   /*
   this->N1 = N1 + 1;  // Periodų skaičius + 1 (gardelei)
   deltaT = (N2 / 252.) / (double)N1;
   double delta = log(1 + i);
   R = exp(delta * deltaT);
   //R = 1 + (pow(1 + i, deltaT) - 1);  // Sudėtinės palūkanos !!!
   */
}
//---------------------------------------------------------------------------
// Sandorio parametrų įvedimas iš laukelių.
//---------------------------------------------------------------------------

void Modelis::IvestiParametrus(double Snul, /* double i, double X,
                               int N1, int N2, */ double ELnS, double sigmaLnS)
{
   TrintiAtminti();

   this->ELnS = ELnS;
   this->sigmaLnS = sigmaLnS;
   this->Snul = Snul;

   PV.kainosNuskaitytos = true;
   PV.parametraiIvesti = true;
   PV.istoriniaiDuomenys = false;
   PV.branduolinisTankis = false;
   PV.dalimisTolygusTankis = false;

   mm->Lines->Add("");
   mm->Lines->Add("Snul = " + AnsiString(Snul));
   mm->Lines->Add("E(LnS) = " + AnsiString(ELnS));
   mm->Lines->Add("sigma(LnS) = " + AnsiString(sigmaLnS));
}
//---------------------------------------------------------------------------
// Sandorio kaina pagal Black & Scholes formulę.
//---------------------------------------------------------------------------

void Modelis::TiksliKaina()
{
   if (PV.kainosNuskaitytos){
      if (PV.parametraiIvesti){
         mm->Lines->Add("_____________________________________________________");
         mm->Lines->Add("Black & Scholes formulė");
         mm->Lines->Add("");
         double kaina = TiksliKaina(Snul, N2);
         mm->Lines->Add("Sandorio kaina: " + FloatToStr(kaina));
      }
      else ShowMessage("Neįvesti sandorio parametrai!");
   }
   else ShowMessage("Aktyvų kainos neįvestos!");
}
//---------------------------------------------------------------------------
// Čia N yra mėnesių skaičius (nebūtinai sveikasis)
//---------------------------------------------------------------------------

double Modelis::TiksliKaina(double Snul, int N)
{
```

```
   double t = N / 252.;
   double delta = log(1 + i);
   double d1 = (log(Snul / X) + (delta + pow(sigmaLnS, 2) / 2.) * t)
         / (sigmaLnS * sqrt(t));
   double d2 = d1 - sigmaLnS * sqrt(t);
   double kaina = 0;
   if (tipas == "Call")
      kaina = Snul * Normal(d1) - X * exp(-delta * t) * Normal(d2);
   if (tipas == "Put")
      kaina = -Snul * Normal(-d1) + X * exp(-delta * t) * Normal(-d2);
   return kaina;
}
//---------------------------------------------------------------------------
// Branduolinio tankio įvertinio sudarymas.
//---------------------------------------------------------------------------

void Modelis::SudarytiBranduoliniTanki(double kernelH)
{
   if (PV.istoriniaiDuomenys){
      if (kernelX){
         delete [] kernelX;
         kernelX = NULL;
      }
      if (kernelP){
         delete [] kernelP;
         kernelP = NULL;
      }
      kernelN = 100;
      //kernelH = pow(kernelN/3., 1/5.) * sqrt(sigmaLnS) * pow(kernelN, -1/5.);
      this->kernelH = kernelH;
      //kernelH = 1;
      mm->Lines->Add("kernelH = " + AnsiString(kernelH));
      kernelX = new double [kernelN];
      kernelP = new double [kernelN];
      double min = Minimum(returns, N4-1);
      double max = Maximum(returns, N4-1);
      // Taking the hard tail in the front and in the back
      while (fbx(returns, N4-1, min, kernelH) > 0.001)
         min -= 0.01;
      while (fbx(returns, N4-1, max, kernelH) > 0.001)
         max += 0.01;
      double dx = (max - min) / (kernelN - 1);
      mm->Lines->Add("Min = " + AnsiString(min) + "\tMax = " + AnsiString(max));
      for (int i = 0; i < kernelN; i++){
         kernelX[i] = min + i*dx;
         kernelP[i] = fbx(returns, N4-1, kernelX[i], kernelH);
         //mm->Lines->Add(AnsiString(kernelX[i]) + "\t" + AnsiString(kernelP[i]));
      }
      //delete [] kernelX;
      //delete [] kernelP;
      PV.branduolinisTankis = true;
      //kernelData = ;
      //kernelH = ;
   }
   else
      ShowMessage("Iš failo nenuskaitytos aktyvo kainos!");
}
//---------------------------------------------------------------------------
// Branduolinio tankio įvertinio vaizdavimas.
//---------------------------------------------------------------------------

void Modelis::VaizduotiBranduoliniTanki()
{
   Form4->BraizytiTrajektorija(kernelX, kernelP, kernelN, "Line", "x", "fbx(x)",

                                       clBlack, "Branduolinis tankio įvertinys");
   Form4->RodytiLegenda(true);
   Form4->Show();
}
//---------------------------------------------------------------------------
// Alternatyvaus tankio vaizdavimas.
//---------------------------------------------------------------------------

void Modelis::VaizduotiAlternatyvuTanki()
{
   int histN = pLinearN-1;
   double * x = new double [histN];
   double * y = new double [histN];
   for (int i = 0; i < histN; i++){
      x[i] = (pLinear[0][i+1] + pLinear[0][i]) / 2;
```

```
      y[i] = pLinear[1][i+1];
   }

   Form4->TrintiGrafika();
   Form4->BraizytiTrajektorija(x, y, histN, "Bar", "x", "Alt(x)",

                                       clYellow, "Alternatyvus tankis");
   Form4->BraizytiTrajektorija(kernelX, kernelP, kernelN, "Line", "x", "fbx(x)",

                                       clBlack, "Branduolinis tankio įvertinys");
   Form4->RodytiLegenda(true);
   Form4->Show();

   delete [] x;
   delete [] y;
}
//---------------------------------------------------------------------------
```

Metodai.h

```
//---------------------------------------------------------------------------
#ifndef MetodaiH
#define MetodaiH
//---------------------------------------------------------------------------
#include "math.h"
#include "Grafikas.h"
//---------------------------------------------------------------------------
//const double PI = 2 * acos(0.0);
// ParkMiller
const long PMa = 16807;
const long PMm = 2147483647;
const long PMq = 127773;
const long PMr = 2836;
//---------------------------------------------------------------------------
class Kintamieji
{
   public:
      int kernelN;
      double kernelH;  // Spartesniam fbx skaičiavimui
      double * kernelData;  // Spartesniam fbx skaičiavimui
      double * kernelX;
      double * kernelP;
};
//---------------------------------------------------------------------------
// Bendrų ir specifinių metodų klasė, kuri bus paveldima.
//---------------------------------------------------------------------------
class Metodai : public Kintamieji
{
   private:
      long PMSeed;
   public:
      TOpenDialog * OpenDialog1;
   public:
      Metodai();
      ~Metodai();
      double Ismoka(double St, double X, AnsiString tipas);
      double ** DoubleMatrica(int m, int n);
      void TrintiDoubleMatrica(double ** M, int m);
      void Kopija(double ** A, double ** B, int m, int n);
      void Daugyba(double ** A, double ** B, double ** C, int m, int n);
      double VektoriausVidurkis(double * M, int m);
      double VektoriausStandartas(double * M, int m, double miu);
      double RandZ();
      double Normal(double x);
      void RodytiMatrica(TMemo * mm, double ** M, int m, int n, int prec);
      AnsiString FailoVardas(AnsiString kelias);
      AnsiString DFN();
      double fbx(double * z, int n, double x, double h);
      double Kx(double x);
      double Minimum(double * x, int n);
      double Maximum(double * x, int n);
      double TrapecijosS(double x1, double x2, double y1, double y2);
      double ** AlternatyvusSK(double * x, double * z, int n, int kiek);
      double ATReiksme(double * x, double * z, int n, double x0);
      double ATInvReiksme(double * x, double * z, int n, double t0);
      double ** Histograma(double * X, int N, int n, bool flag = false);
      //ptrdiff_t myrandom (ptrdiff_t i);
      long longParkMiller();
      double doubleParkMiller();
};
```

```
//---------------------------------------------------------------------------
#endif
```

Metodai.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Metodai.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
//---------------------------------------------------------------------------

Metodai::Metodai()
{
   PMSeed = 1;
}
//---------------------------------------------------------------------------

Metodai::~Metodai()
{
}
//---------------------------------------------------------------------------
// Sandorio išmoka. Call: max(St - X, 0). Put: max(X - St, 0).
//---------------------------------------------------------------------------

double Metodai::Ismoka(double St, double X, AnsiString tipas)
{
   if (tipas == "Call")
      if (St <= X) return 0;
         else return St - X;
   if (tipas == "Put")
      if (St >= X) return 0;
         else return X - St;
   return St;
}
//---------------------------------------------------------------------------
// Grąžina m x n double tipo dinaminę matricą.
//---------------------------------------------------------------------------

double ** Metodai::DoubleMatrica(int m, int n)
{
   double ** M = new double * [m];
   for (int i = 0; i < m; i++){
      M[i] = new double [n];
      for (int j = 0; j < n; j++)
         M[i][j] = 0;
   }
   return M;
}
//---------------------------------------------------------------------------
// Trina m x n double tipo dinaminę matricą (atlaisvinama atmintis).
//---------------------------------------------------------------------------

void Metodai::TrintiDoubleMatrica(double ** M, int m)
{
   if (M){
      for (int i = 0; i < m; i++)
         delete [] M[i];
      delete [] M;
      // M = NULL;  // Nebūtų grąžinamas !!!
   }
}
//---------------------------------------------------------------------------
// Matrica A yra kopijuojama į matricą B.
//---------------------------------------------------------------------------

void Metodai::Kopija(double ** A, double ** B, int m, int n)
{
   for (int i = 0; i < m; i++)
      for (int j = 0; j < n; j++)
         B[i][j] = A[i][j];
}
//---------------------------------------------------------------------------
// Atliekama matricų daugyba A * B = C.
//---------------------------------------------------------------------------

void Metodai::Daugyba(double ** A, double ** B, double ** C, int m, int n)
{
   for (int i = 0; i < m; i++){
```

```
      for (int j = 0; j < n; j++){
         double suma = 0;
         for (int k = 0; k < m; k++)
            suma += A[i][k] * B[k][j];
         C[i][j] = suma;
      }
   }
}
//---------------------------------------------------------------------------
// Vektoriaus vidurkis.
//---------------------------------------------------------------------------

double Metodai::VektoriausVidurkis(double * M, int m)
{
   double rez = 0;
   if (M){
      double sum = 0;
      for (int i = 0; i < m; i++)
         sum += M[i];
      rez = sum / m;
   }
   return rez;
}
//---------------------------------------------------------------------------
// Vektoriaus standartas.
//---------------------------------------------------------------------------

double Metodai::VektoriausStandartas(double * M, int m, double miu)
{
   double rez = 0;
   if (M){
      double sum = 0;
      for (int i = 0; i < m; i++)
         sum += (M[i] - miu)*(M[i] - miu);
      rez = sqrt(sum / (m - 1));
   }
   return rez;
}
//---------------------------------------------------------------------------
// Grąžina atsitiktinį skaičių ~ N(0, 1).
//---------------------------------------------------------------------------

double Metodai::RandZ()
{
   double Z1 = (double)rand() / ((double)(RAND_MAX));
   double Z2 = (double)rand() / ((double)(RAND_MAX));
   double Z3 = (double)rand() / ((double)(RAND_MAX));
   double Z4 = (double)rand() / ((double)(RAND_MAX));
   double Z5 = (double)rand() / ((double)(RAND_MAX));
   double Z6 = (double)rand() / ((double)(RAND_MAX));

   double rnd = ((Z1 + Z2 + Z3 + Z4 + Z5 + Z6) - 3) * sqrt(2);
   return rnd;
}
//---------------------------------------------------------------------------
// Abromowitz and Stegun aproksimacija normaliajam skirstiniui (CDF).
//---------------------------------------------------------------------------

double Metodai::Normal(double x)
{
   double b1 =  0.319381530;
   double b2 = -0.356563782;
   double b3 =  1.781477937;
   double b4 = -1.821255978;
   double b5 =  1.330274429;
   double p  =  0.2316419;
   double c  =  0.39894228;

   if(x >= 0.0){
      double t = 1.0 / ( 1.0 + p * x );
      return (1.0 - c * exp( -x * x / 2.0 ) * t *
            ( t *( t * ( t * ( t * b5 + b4 ) + b3 ) + b2 ) + b1 ));
   }
   else{
      double t = 1.0 / ( 1.0 - p * x );
      return ( c * exp( -x * x / 2.0 ) * t *
            ( t *( t * ( t * ( t * b5 + b4 ) + b3 ) + b2 ) + b1 ));
   }
}
//---------------------------------------------------------------------------
```

```
// Metodas matricai TMemo lange atspausdinti.
//---------------------------------------------------------------------------

void Metodai::RodytiMatrica(TMemo * mm, double ** M, int m, int n, int prec)
{
   AnsiString line = "";
   AnsiString line2 = "+";
   AnsiString skaicius = "";
   for (int i = 0; i < n; i++){
      if (i == n - 1)
         line2 += "+--";
      line2 += "----------";
   }
   line2 += "+";
   mm->Lines->Add(line2);
   for (int i = 0; i < m; i++){
      line = "|";
      for (int j = 0; j < n; j++){
         //skaicius = FloatToStrF(M[i][j], ffGeneral, prec, 7);
         if (prec == 2) skaicius = FormatFloat("0.00", M[i][j]);
         if (prec == 3) skaicius = FormatFloat("0.000", M[i][j]);
         while (skaicius.Length() < 10)
            skaicius = " " + skaicius;
         if (j == n - 1)
            line += "| ";
         line += skaicius;
      }
      mm->Lines->Add(line + " |");
      line = "";
   }
   mm->Lines->Add(line2);
}
//---------------------------------------------------------------------------
// Iškviečiamas dialogas duomenų failui nurodyti.
//---------------------------------------------------------------------------

AnsiString Metodai::DFN()
{
   if (OpenDialog1->Execute() && FileExists(OpenDialog1->FileName)){
      return OpenDialog1->FileName;
   }
   else{
      return "";
   }
}
//---------------------------------------------------------------------------
// Iš pilno failo adreso išskiriamas tik jo vardas.
//---------------------------------------------------------------------------

AnsiString Metodai::FailoVardas(AnsiString kelias)
{
   AnsiString fv = kelias;
   int slashas = kelias.Pos("\\");
   bool yraSlashu = false;
   if (slashas != 0) yraSlashu = true;
   while (yraSlashu){
      slashas = fv.Pos("\\");
      yraSlashu = false;
      if (slashas != 0){
         yraSlashu = true;
         fv = fv.SubString(slashas + 1, fv.Length() - 3);
      }
   }
   return fv;
}
//---------------------------------------------------------------------------
// Branduolinis tankis, z - imties reikšmių masyvas, n - jo elementų skaičius,
// h - branduolio plotis,
// x - taškas, kuriame skaičiuojama branduolio funkcijos reikšmė.
//---------------------------------------------------------------------------

double Metodai::fbx(double * z, int n, double x, double h)
{
   double suma = 0;
   for (int i = 0; i < n; i++)
      suma = suma + Kx((x - z[i])/h) / h;
   return suma / n;
}
//---------------------------------------------------------------------------
// Branduolio funkcija, h - jo plotis.
```

```
//---------------------------------------------------------------------------

double Metodai::Kx(double x)
{
   // Gauso branduolys
   double sigma = 1;
   double PI = 2 * acos(0.0);
   return exp(-x*x / (2*sigma*sigma)) / sqrt(2*PI);
}
//---------------------------------------------------------------------------
// Masyvo minimunas.
//---------------------------------------------------------------------------

double Metodai::Minimum(double * x, int n)
{
   double minimum = 99999;
   for (int i = 0; i < n; i++)
      if (minimum > x[i])
         minimum = x[i];
   return minimum;
}
//---------------------------------------------------------------------------
// Masyvo maksimumas.
//---------------------------------------------------------------------------

double Metodai::Maximum(double * x, int n)
{
   double maximum = -99999;
   for (int i = 0; i < n; i++)
      if (maximum < x[i])
         maximum = x[i];
   return maximum;
}
//---------------------------------------------------------------------------
// Trapecojos ploto skaičiavimas.
//---------------------------------------------------------------------------

double Metodai::TrapecijosS(double x1, double x2, double y1, double y2)
{
             double aukstis = y1;
   if (y2 < y1)
      aukstis = y2;
   double plotis = x2 - x1;
   double aukstisT = fabs(y2 - y1);
   double staciakampioS = plotis * aukstis;
   double trikampioS = (aukstisT * plotis) / 2;
   return staciakampioS + trikampioS;
}
//---------------------------------------------------------------------------
// Alternatyvaus tankio (dalimis tolygusis skirstinys) konstravimas.
// x - neparametrinio tankio įverčio argumentai.
// z - neparametrinio tankio įverčio reikšmės.
// n - įvertintų taškų kiekis.
// kiek - dalimis tolygaus skirstinio intervalų skaičius + 1.
// Čia x yra intervalų galai: x[0], x[1], ..., x[n-1].
//---------------------------------------------------------------------------

double ** Metodai::AlternatyvusSK(double * x, double * z, int n, int kiek)
{
             if (kiek > 1){
                          double ** alternative = DoubleMatrica(3, kiek);
                          //alternatyvaX(1) = xx(1);
                          //alternatyvaPDF(1) = 0;
                          alternative[0][0] = x[0];  // First point of PDF estimate
                          alternative[1][0] = 0;  // PDF starts with 0
                          int * index = new int [kiek];
                          index[0] = 0;
      alternative[0][0] = x[0];
                          for (int i = 1; i < kiek; i++){  //i = 2:kiek,
                                       index[i] = (i*(n-1))/(kiek-1);
         //while (index[i] >= n)
            //index[i]--;
                                       alternative[0][i] = x[index[i]];
                                       // START: Plotas po tankiu intervale
                                       double trapecijuS = 0;
                                       for (int j = index[i-1]; j < index[i]; j++)
                                                    trapecijuS  +=  TrapecijosS(x[j],  x[j+1],  z[j],
z[j+1]);
                                       alternative[1][i]  =  trapecijuS  /  (x[index[i]]  -  x[index[i-
1]]);
```

```
         //ShowMessage(alternative[0][i]);
                                        // END: Plotas po tankiu intervale
                                        if (i == 1)
                                                alternative[2][i] = trapecijuS;
                                        else{
                                                if (i == kiek - 1)
                                                        alternative[2][i] = 1;
                                                else
                                                        alternative[2][i]                    =
alternative[2][i-1] + trapecijuS;
                                        }
                           }
      /*
      // Patikrinimui
      AnsiString line1 = "";
      AnsiString line2 = "";
      AnsiString line3 = "";
      for (int i = 0; i < kiek; i++){
         line1 += " " + FloatToStrF(alternative[0][i], ffFixed, 8, 4);
         line2 += " " + FloatToStrF(alternative[1][i], ffFixed, 8, 4);
         line3 += " " + FloatToStrF(alternative[2][i], ffFixed, 8, 4);
      }
      ShowMessage(line1 + "\n" + line2 + "\n" + line3);
      */
                           return alternative;
              }
              else
                           return NULL;
}
//---------------------------------------------------------------------------
// Alternatyvaus tankio reikšmė taške.
//---------------------------------------------------------------------------

double Metodai::ATReiksme(double * x, double * z, int n, double x0)
{
   int ind = 1; // Desinysis indeksas
   while ((ind <= n) && (x[ind] < x0))
      ind++;
   if (ind > n)
      ind = n;
   return z[ind];
}
//---------------------------------------------------------------------------
// Alternatyvaus pasiskirstymo atvirkštinės funkcijos reikšmė taške.
//---------------------------------------------------------------------------

double Metodai::ATInvReiksme(double * x, double * z, int n, double t0)
{
   int ind = 1; // Dešinysis indeksas !!!
   while (z[ind] < t0)
      ind++;

   // (x-x1)/(x2-x1) = (y-y1)/(y2-y1) => x = (x2-x1)*(y-y1)/(y2-y1) + x1
   double x2 = x[ind];
   double x1 = x[ind-1];
   double y2 = z[ind];
   double y1 = z[ind-1];

   return (x2-x1)*(t0-y1)/(y2-y1) + x1;
}
//---------------------------------------------------------------------------
// hist [x0; xn]
// n - intervalų skaičius + 1.
// flag = false - rezultatas bus dešiniosios intervalų ribos ir tikimybės.
// flag = true - rezultatas bus intervalų centrai ir tikimybės.
//---------------------------------------------------------------------------

double ** Metodai::Histograma(double * X, int N, int n, bool flag)
{
   double ** hist = DoubleMatrica(2, n);
   double x0 = Minimum(X, N);
   //if (xn = NULL)
   double xn = Maximum(X, N);
   for (int i = 0; i < n; i++)
      hist[1][i] = 0;
   int dazniuSuma = 0;
   for (int i = 0; i < N; i++){
      /*if (flag){
         for (int j = 1; j <= n; j++){
                double riba = x0 + (xn - x0) * j/(double)n;
```

```
            if (j == n)
               riba = xn;
                 if (X[i] <= riba){
                 hist[1][j-1] += 1;
               hist[0][j-1] = riba;
               dazniuSuma++;
               j = n + 1;
            }
         }
      }
      else{  */
         for (int j = 1; j <= n; j++){
                double riba = x0 + (xn - x0) * j/(double)n;
            if (j == n)
               riba = xn;
            if (flag){
               double riba2 = x0 + (xn - x0) * (j-1)/(double)n;
               hist[0][j-1] = (riba + riba2) / 2.;
            }
            else
               hist[0][j-1] = riba;
                 if (X[i] <= riba){
                 hist[1][j-1] += 1;
               dazniuSuma++;
               j = n + 1;
            }
         }
      //}
   }
   //ShowMessage("Dažnių suma: " + IntToStr(dazniuSuma));
   for (int i = 0; i < n; i++)
                           hist[1][i] /= ((xn - x0) * (double)N/(double)n);
   /*
   if (flag){
      int histN = pLinearN-1;
      double * x = new double [histN];
      double * y = new double [histN];
      double ** hist2 = DoubleMatrica(2, n-1);
      for (int i = 0; i < histN; i++){
         x[i] = (pLinear[0][i+1] + pLinear[0][i]) / 2;
         y[i] = pLinear[1][i+1];
      }
   }
   */
   return hist;
}
//---------------------------------------------------------------------------
// ParkMiller atsitiktinių skaičių generatorius.
//---------------------------------------------------------------------------

long Metodai::longParkMiller()  // [1; m-1]
{
   long k;
   k = PMSeed / PMq;
   PMSeed = PMa * (PMSeed-k*PMq) - PMr*k;
   if (PMSeed < 0)
      PMSeed += PMm;
   return PMSeed;
}
//---------------------------------------------------------------------------
// ParkMiller atsitiktinių skaičių generatorius.
//---------------------------------------------------------------------------

double Metodai::doubleParkMiller()  // [1; m-1]
{
   return longParkMiller()/(double)PMm;
}
//---------------------------------------------------------------------------
```

2 priedas. Pagrindinės MATLAB programos kodas

```
clear all;
clc;


failai{1} = 'Boeing Company (BO)';
failai{2} = 'Tata Motors Ltd. (TTM)';
failai{3} = 'Apple Inc. (AAPL)';
failai{4} = 'Hewlett-Packard Company (HPQ)';
failai{5} = 'Advanced Micro Devices (AMD)';
```

```
failai{6} = 'Nokia Corporation (NOK)';
failai{7} = 'Tesco Corporation (TESO)';
failai{8} = 'SAINSBURY (SBRY.L)';
failai{9} = 'Yahoo! Inc. (YHOO)';
failai{10} = 'Yanzhou Coal Mining Co. Ltd. (YZC)';
failai{11} = 'Fiktyvus3 (yra normalumas)';
failoID = 11;

%-------------------------------------------------------------------------
% Programos kintamieji
periodai = 30; % bendras periodu skaicius (252)
dienosN = 2; % periodu dienai skaicius
trajektorijos = 500;
kiek = 4; % Dalimis tolygiojo skirstinio intervalu skaicius + 1

% Nuskaitomos kainos
SkaitytiKainas;

% Randamos pelno normos
time10 = cputime;
%-------------------------------------------------------------------------
for i = 1:N - 1,
    graza(i) = (kainos(i+1) - kainos(i)) / kainos(i);
    % Modelioujant tada Sn1 = rn*Sn + Sn
    grazaLN(i) = log(kainos(i+1) / kainos(i));
end;
dlmwrite(strcat('Returns/', failai{failoID}, '.txt'), graza');
figure(5); clf;
%set(gca, 'Position', [0 0 1 1]);
bar(graza);
title({'Akciju kainu grazos', failai{failoID}});
%-------------------------------------------------------------------------
time11 = cputime;

% Grazu skaitiniu charakteristiku radimas
time20 = cputime;
%-------------------------------------------------------------------------
miu = mean(graza) * 252;
sigma = sqrt(var(graza) * 252);
miuLN = mean(grazaLN) * 252;
sigmaLN = sqrt(var(grazaLN) * 252);
disp(['   Metinis miu = ' num2str(miu)]);
disp(['   Metinis sigma = ' num2str(sigma)]);
disp(['   Metinis miuLN = ' num2str(miuLN)]);
disp(['   Metinis sigmaLN = ' num2str(sigmaLN)]);
%-------------------------------------------------------------------------
time21 = cputime;

% Grazu logaritmu normalizavimas ir normalumo tikrinimas
time30 = cputime;
%-------------------------------------------------------------------------
% Normalizavimas
for i = 1:N - 1,
    grazaNLN(i) = (grazaLN(i) - miuLN / 252) / (sigmaLN * sqrt(1 / 252));
end;
figure(2); clf; set(gcf,'Color',[1 1 1]);
range1 = [-3:0.5:3];
range2 = [-3:0.1:3];
H = hist(grazaNLN, range1) ./ ((N - 1) * 0.5);
hold on;
bar(range1, H);
plot(range2, normpdf(range2, 0, 1), 'r');
hold off;
%title({'Histogram of normalized log of returns', failai{failoID}});
%title('Normalized R_t = ln(S_t/S_{t-1})');
xlabel('R_t');
ylabel('f(R_t)');
% Normalumo tikrinimas
[h temp D p] = kstest(grazaNLN, [], 0.01);
disp(['   p = ' num2str(p)]);
disp(['   Kolmogorovo statistika D = ' num2str(D)]);
if h == 0,
    disp('   Hipoteze apie grazu normaluma priimta.');
else
    disp('   Hipoteze apie grazu normaluma atmesta.');
end;
%-------------------------------------------------------------------------
```

```
time31 = cputime;

% Pelno normu skirstinio branduolinis ivertis
time40 = cputime;
% % Grazu normalizavimas
% graza = (graza - mean(graza))/std(graza);
%-------------------------------------------------------------------------
% Optimalaus branduolio plocio ieskojimas
d = 1;
%h = power((N/(2+d)), 1/(4+d)) * sqrt(sigma) * power(N, -1/(4+d));
h = 0.03;
%h = 0.3;
disp(['   Optimalus h = ' num2str(h)]);
% Branduolinio ivercio sudarymas
figure(6); clf;
subplot(1, 2, 1);
mz = min(graza) - 0.05;
Mz = max(graza) + 0.05;
dx = 0.005;
ind = 1;
for k = mz:dx:Mz,
    xx(ind) = k;
    zz(ind) = fbx(graza, xx(ind), h);
    ind = ind + 1;
end;
plot(xx, zz, 'b');
title(['Branduolinis tankio ivertis (h = ' num2str(h) ')']);
% Branduolinio tankio ivercio, normaliojo skirstinio
% ir grazu histogramos palyginimas
dx = (Mz-mz)/20;
range = mz:dx:Mz;
range2 = mz:(dx/5):Mz;
H = hist(graza, range) ./ ((N - 1) * dx);
subplot(1, 2, 2);
hold on;
bar(range, H);
%plot(range2, normpdf(range2, miu/252, sigma*sqrt(1 / 252)), 'r--');
plot(range2, normpdf(range2, mean(graza), std(graza)), 'r--');
plot(xx, zz, 'g');
hold off;
axis tight;
%-------------------------------------------------------------------------
time41 = cputime;

% Alternatyvaus skirstinio konstravimas
time50 = cputime;
%-------------------------------------------------------------------------
nn = length(zz);
[alternatyvaX alternatyvaPDF alternatyvaCDF] = Alternatyva(xx, zz, kiek);
figure(7); clf; subplot(1, 2, 1);
hold on;
plot(xx, zz, 'g--');
% Dalimis tolygiojo tankio vaizdavimas
for i = 1:kiek,
    if i < kiek,
        plot([alternatyvaX(i) alternatyvaX(i)], [alternatyvaPDF(i) alternatyvaPDF(i+1)]);
        plot([alternatyvaX(i) alternatyvaX(i+1)], [alternatyvaPDF(i+1) alternatyvaPDF(i+1)]);
        plot(alternatyvaX(i), alternatyvaPDF(i), 'bo');
        plot(alternatyvaX(i), alternatyvaPDF(i+1), 'bo', 'MarkerFaceColor', 'k');
    else
        plot([alternatyvaX(i) alternatyvaX(i)], [alternatyvaPDF(i) 0]);
        plot(alternatyvaX(i), alternatyvaPDF(i), 'bo');
        plot(alternatyvaX(i), 0, 'bo', 'MarkerFaceColor', 'k');
    end;
end;
axis([min(xx) max(xx) 0 max(zz)]);
xlabel('R_t');
ylabel('f(R_t)');
title('Alternatyvus tankis');
hold off;
subplot(1, 2, 2);
hold on;
plot(alternatyvaX, alternatyvaCDF);
xlabel('R_t');
ylabel('F(R_t)');
title('Alternatyvi pasiskirstymo funkcija');
% Pavyzdiniu reiksmiu generavimas
rnN = trajektorijos*periodai;
rnx = zeros(rnN, 1);
```

```
for i = 1:rnN,
    rn = rand();
    ind = 2; % Desinysis indeksas
    while alternatyvaCDF(ind) < rn,
        ind = ind + 1;
    end;
    % (x-x1)/(x2-x1) = (y-y1)/(y2-y1) => x = (x2-x1)*(y-y1)/(y2-y1) + x1
    x2 = alternatyvaX(ind);
    x1 = alternatyvaX(ind-1);
    y2 = alternatyvaCDF(ind);
    y1 = alternatyvaCDF(ind-1);
    y = rn;
    rnx(i) = (x2-x1)*(y-y1)/(y2-y1) + x1;
    if i <= 20,
        plot([alternatyvaX(1) rnx(i)], [y y], 'r--');
        plot([rnx(i) rnx(i)], [y 0], 'r--');
    end;
end;
axis([min(xx) max(xx) 0 1]);
hold off;
% Alternatyvaus skirstinio ir branduolinio tankio ivercio palyginimas
dx = (Mz-mz)/30;
range = mz:dx:Mz;
H = hist(rnx, range) ./ (rnN * dx);
figure(8); clf;
hold on;
bar(range, H);
plot(xx, zz, 'g--');
% Alternatyvus tankis
for i = 1:kiek,
    if i < kiek,
        plot([alternatyvaX(i) alternatyvaX(i)], [alternatyvaPDF(i) alternatyvaPDF(i+1)]);
        plot([alternatyvaX(i) alternatyvaX(i+1)], [alternatyvaPDF(i+1) alternatyvaPDF(i+1)]);
        plot(alternatyvaX(i), alternatyvaPDF(i), 'bo');
        plot(alternatyvaX(i), alternatyvaPDF(i+1), 'bo', 'MarkerFaceColor', 'k');
    else
        plot([alternatyvaX(i) alternatyvaX(i)], [alternatyvaPDF(i) 0]);
        plot(alternatyvaX(i), alternatyvaPDF(i), 'bo');
        plot(alternatyvaX(i), 0, 'bo', 'MarkerFaceColor', 'k');
    end;
end;
xlabel('R_t');
ylabel('f(R_t)');
title('Alternatyvus tankis');
hold off;
% Reiksmiu pagal MCMC generavimas
MCMC = zeros(rnN, 1);
MCMC(1) = rnx(1);
for i = 2:rnN,
    U1 = rand();
    % proposal
    xi1 = rnx(i); % Independence sampler
    % acceptance
    skaitiklis = fbx(graza, xi1, h) * FunctionValue2(alternatyvaX, alternatyvaPDF, MCMC(i-1));
    vardiklis = fbx(graza, MCMC(i-1), h) * FunctionValue2(alternatyvaX, alternatyvaPDF, xi1);
    if vardiklis == 0,
        vardiklis = 0.0000000000000001;
    end;
    alfa = skaitiklis / vardiklis;
    if alfa > 1,
        alfa = 1;
    end;
    if rand() <= alfa,
        MCMC(i) = xi1;
    else
        MCMC(i) = MCMC(i-1);
    end;
    %disp(['Proposal = ' num2str(xi1) ' alfa = ' num2str(alfa) ' x(i) = ' num2str(MCMC(i))]);
end;
%-------------------------------------------------------------------------
time51 = cputime;

% MCMC ir branduolinio tankio palyginimas
time60 = cputime;
%-------------------------------------------------------------------------
dx = (Mz-mz)/30;
range = mz:dx:Mz;
H = hist(MCMC, range) ./ (rnN * dx);
figure(9); clf;
```

```
hold on;
bar(range, H);
plot(xx, zz, 'g--');
xlabel('R_t');
ylabel('f(R_t)');
title('MCMC ir branduolinis tankio ivertis');
hold off;
%--------------------------------------------------------------------------
time61 = cputime;

% MCMC sumaisymas
time70 = cputime;
%--------------------------------------------------------------------------
position = randperm(rnN);
for i = 1:rnN,
    MCMC2(i) = MCMC(position(i));
end;
MCMC = MCMC2;
%--------------------------------------------------------------------------
time71 = cputime;

% Daugelio kainu trajektoriju generavimas ir vidutines kainu
% trajektorijos radimas
time80 = cputime;
%--------------------------------------------------------------------------
rnx = (MCMC2 - mean(MCMC2))/std(MCMC2);
% rnx = MCMC2;
S0 = kainos(N);
deltat = (1 / 252) / dienosN;
for j = 1:periodai,
    laikas(j) = deltat * (j - 1);
end;

% START: MC
    StC = zeros(trajektorijos, periodai);
    StVidC = zeros(periodai, 1);
    for i = 1:trajektorijos,
        StC(i, 1) = S0;
        for j = 2:periodai,
            Z = normrnd(0, 1);
            StC(i, j) = StC(i, j - 1) * (1 + miu * deltat + sigma * Z * sqrt(deltat));
        end;
    end;
    % Vidutine trajektorija
    for i = 1:periodai,
        StVidC(i) = mean(StC(:, i));
    end;
% END: MC

% START: MCMC
    St = zeros(trajektorijos, periodai);
    StVid = zeros(periodai, 1);
    for i = 1:trajektorijos,
        St(i, 1) = S0;
        for j = 2:periodai,
            %St(i, j) = St(i, j - 1) * (1 + rnx((i-1)*periodai+(j-1)));
            % The new technique
            Z = rnx((i-1)*periodai+(j-1)); % Buvo imami a.d. is rnx()!!!
            St(i, j) = St(i, j - 1) * (1 + miu * deltat + sigma * Z * sqrt(deltat));
            % MINTIS: miu + sigma * Z + f(kurtosis, Z) + ...
        end;
    end;
    % Vidutine trajektorija
    for i = 1:periodai,
        StVid(i) = mean(St(:, i));
    end;
% END: MCMC

% Trajektoriju vaizdavimas
figure(3); clf; set(gcf,'Color',[1 1 1])
subplot(2, 1, 2);
hold on;
%trajektorijos2 = min(periodai, size(kainos, 1));
laikasBEFORE = sort(randperm(N)) ./ 252;
plot(laikasBEFORE, kainos', 'b');
laikasAFTER = laikas + laikasBEFORE(N);
for i = 1:trajektorijos,
```

```
    plot(laikasAFTER, St(i, :), 'r');
end;
plot(laikasAFTER, StVid, 'k');
title([num2str(trajektorijos) ' MCMC trajectories']);
xlabel('t');
ylabel('S_t');
%legend('Duomenys', 'Prognozes', -1);
hold off;
subplot(2, 1, 1);
hold on;
plot(laikasBEFORE, kainos', 'b');
for i = 1:trajektorijos,
    plot(laikasAFTER, StC(i, :), 'r');
end;
plot(laikasAFTER, StVidC, 'k');
title([num2str(trajektorijos) ' Monte Carlo trajectories']);
xlabel('t');
ylabel('S_t');
hold off;
set(gcf, 'PaperPositionMode', 'auto'); print(gcf, '-dmeta', 'Images/3.wmf');
%-------------------------------------------------------------------------
time81 = cputime;

% Opciono ikainojimas
time90 = cputime;
%-------------------------------------------------------------------------
X = 40; % Ivykdymo kaina
r = 0.05; % Metine palukanu norma
% Randamos opciono vertes sandorio ivykdymo metu
Cpab = zeros(trajektorijos, 1);
for i = 1:trajektorijos,
    Cpab(i) = max([0 (St(i, periodai)-X)]);
end;
CpabM = mean(Cpab);
C = CpabM / power(1 + r, periodai/252);
disp(['   S0 = ' num2str(S0)]);
disp(['   X = ' num2str(X)]);
disp(['   r = ' num2str(r)]);
disp(['   CpabM = ' num2str(CpabM)]);
disp(['   C = ' num2str(C)]);
[Ccall, Cput] = blsprice(S0, X, log(1+r), (periodai/2)/252, sigmaLN);
disp(['   BS = ' num2str(Ccall)]);
disp(['   santykine paklaida = ' num2str((C-Ccall)/Ccall)]);
%-------------------------------------------------------------------------
time91 = cputime;

disp(['   -------        Pabaiga         -------']);
time100 = cputime;
figure(2); set(gcf, 'PaperPositionMode', 'auto'); print(gcf, '-dmeta', 'Images/2.wmf');
%figure(4); set(gcf, 'PaperPositionMode', 'auto'); print(gcf, '-dmeta', 'Images/4.wmf');
figure(5); set(gcf, 'PaperPositionMode', 'auto'); print(gcf, '-dmeta', 'Images/5.wmf');
figure(6); set(gcf, 'PaperPositionMode', 'auto'); print(gcf, '-dmeta', 'Images/6.wmf');
figure(7); set(gcf, 'PaperPositionMode', 'auto'); print(gcf, '-dmeta', 'Images/7.wmf');
figure(8); set(gcf, 'PaperPositionMode', 'auto'); print(gcf, '-dmeta', 'Images/8.wmf');
figure(9); set(gcf, 'PaperPositionMode', 'auto'); print(gcf, '-dmeta', 'Images/9.wmf');
time101 = cputime;

% Informacija apie skaiciavimu laika
%-------------------------------------------------------------------------
disp(['   Kainu skaitymas:              ', num2str(time01-time00), ' s.']);
disp(['   Pelno normu radimas:          ', num2str(time11-time10), ' s.']);
disp(['   Grazu skaitines ch.:          ', num2str(time21-time20), ' s.']);
disp(['   Grazu normalumo tikrinimas:   ', num2str(time31-time30), ' s.']);
disp(['   Branduolinis tankis:          ', num2str(time41-time40), ' s.']);
disp(['   Alternatyvus sk. ir MCMC:     ', num2str(time51-time50), ' s.']);
disp(['   MCMC ir br.. t. palyginimas:  ', num2str(time61-time60), ' s.']);
disp(['   MCMC maisymas:                ', num2str(time71-time70), ' s.']);
disp(['   Trajektorijos:                ', num2str(time81-time80), ' s.']);
disp(['   Opciono ikainojimas:          ', num2str(time91-time90), ' s.']);
disp(['   Grafiku saugojimas:           ', num2str(time101-time100), ' s.']);
```

3 priedas. Papildomų MATLAB programų kodai

Fbx.m

```
function rez = fbx(z, x, h)
suma = 0;
n = length(z);
for i = 1:n
    suma = suma + Khx(x - z(i), h);
end;
rez = suma / n;
```

Khx.m

```
function ans = Khx(x, h)
ans = Kx(x/h) / h;
```

Kx.m

```
function ans = Kx(x)
sigma = 1;
ans = exp(-x*x / (2 * sigma * sigma)) / sqrt(2*pi);
```

SkaitytiKainas.m

```
time00 = cputime;
%-------------------------------------------------------------------------
disp(['   -------  ' failai{failoID} '   -------']);
disp(['   -------  Istoriniai duomenys   -------']);
kainos = textread(strcat('Prices/', failai{failoID}, '.txt'), '%f');
N = size(kainos, 1);
% Kainos pateiktos nuo naujausios, perrikiuojama
% kainos = flipud(kainos);
figure(1); clf; set(gcf,'Color',[1 1 1]);
plot(kainos);
title(failai{failoID});
xlabel('t (starting from 2010-01-04)');
ylabel('S_t');
set(gcf, 'PaperPositionMode', 'auto'); print(gcf, '-dmeta', 'Images/1.wmf');
%-------------------------------------------------------------------------
time01 = cputime;
```

TrapecijosS.m

```
function rez = TrapecijosS(x1, x2, y1, y2)
    aukstis = min([y2 y1]);
    plotis = x2 - x1;
    aukstisT = abs(y2 - y1);
    staciakampioS = plotis * aukstis;
    trikampioS = (aukstisT * plotis) / 2;
    rez = staciakampioS + trikampioS;
```

Alternatyva.m

```
function [alternatyvaX, alternatyvaPDF, alternatyvaCDF] = Alternatyva(xx, zz, kiek)


    nn = length(zz);
    alternatyvaPDF(1) = 0;
    alternatyvaX(1) = xx(1);
    index(1) = 1;
    for i = 2:kiek,
        index(i) = floor((i-1)*nn/(kiek-1));
        alternatyvaX(i) = xx(index(i));
        % Suintegruojamas tankis tarp isrinktu tasku
        trapecijuS = 0;
        for j = index(i-1):index(i)-1,
            trapecijuS = trapecijuS + TrapecijosS(xx(j), xx(j+1), zz(j), zz(j+1));
        end;
        alternatyvaPDF(i) = trapecijuS / (xx(index(i)) - xx(index(i-1)));
        if i == 2,
            alternatyvaCDF(i) = trapecijuS;
        else
            if i == kiek,
                alternatyvaCDF(i) = 1;
            else
                alternatyvaCDF(i) = alternatyvaCDF(i-1) + trapecijuS;
            end;
        end;
    end;
```

FunctionValue2.m

```
function rez = FunctionValue2(xxx, yyy, x)
    ind = 1; % Desinysis indeksas
    nnn = length(xxx);
    while (ind <= nnn) && (xxx(ind) < x),
        ind = ind + 1;
    end;
    if ind > nnn,
        ind = nnn;
    end;
    rez = yyy(ind);
```