KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Virginijus Pampikas

Paciento valdomos analgezijos vaistų pompos algoritmų tyrimas

Magistro darbas

Darbo vadovas
prof. habil. dr. H. Pranevičius

Kaunas, 2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Virginijus Pampikas

Paciento valdomos analgezijos vaistų pompos algoritmų tyrimas

Magistro darbas

Recenzentas
prof. dr. E. Bareiša
2011-05 -30

Vadovas
prof. habil. dr. H. Pranevičius
2011-05-27

Atliko
IFM-9/2 gr. stud.
Virginijus Pampikas
2011-05-27

Kaunas, 2011

Santrauka

Šiame darbe atliekamas paciento valdomos analgezijos vaistų pompos galimų veikimo algoritmų (paprastojo ir infuzinio) tyrimas, siekiant nustatyti, kuris algoritmas yra veiksmingesnis.

Tyrimui atlikti, naudojantis hibridinių sistemų bei kvantuotų sistemos būsenų modeliais, buvo sudarytas agregatinis imitacinis paciento kontroliuojamos analgezijos modelis (PKA). Imitacinio modelio komponentai buvo aprašyti naudojant PLA formalizavimo metodą. Siekiant įgyvendinti imitacinį PKA modelį, buvo panaudota agregatinių sistemų modeliavimo programa (autoriai: Virginijus Pampikas ir Donatas Urbas).

Tyrimo metu, naudojant imitacinį PKA modelį, buvo atliekami imitaciniai eksperimentai, kurių metu buvo stebimas analgezijos vaistų koncentracijos kitimas paciento organizme. Lyginant gautuosius rezultatus, buvo siekiama nustatyti tiriamos vaistų pompos algoritmų pranašumus ir trūkumus vienas kito atžvilgiu.

Summary

This MA thesis contains an analysis of possible operation of algorithms (conventional and infusion) of analgesics pump controlled by a patient. The aim is to find out which algorithm is more effective.

In order to perform the analysis, models of hybrid systems and quantized state system were used and a model of an aggregate imitable patient-controlled analgesia (PCA) was produced. The PLA formalization model was used to describe the components of the simulation model. In order to implement the simulation PCA model, the program for aggregate systems modelling was used (authors: Virginijus Pampikas and Donatas Urbas).

During the analysis, the simulation PCA model was used and the simulation experiments were carried out. During these experiments the change of analgesics concentration in a patient's body was observed. The results were compared in order to determine the pros and cons of the algorithms of the analysed drug pump.

Turinys

Iliustracijų turinys

Lentelių turinys

1. ĮVADAS

Šiandieniniame pasaulyje, informacinių technologijų (IT) skvarba siekia beveik visas žmonių buities bei veiklos sritis. Ne išimtis ir medicina, kur naujausių informacinių technologijų taikymas padeda taupyti išlaidas, suteikia pacientams bei gydytojams daugiau komforto, neretai gelbsti gyvybes. Siekiant minėtųjų rezultatų, medicinos srityje ypatingai svarbu kurti patikimą medicininę įrangą ir gydymo metodus. Šiuo atveju, būtina tinkamai įvertinti galimus IT sprendimų taikymus bei jų taikymo pasekmes. Tam tikslui yra kuriami imitaciniai modeliai, kurie padeda atlikti skaičiavimus ir patikrinti imituojamos sistemos teikiamą naudą bei jos veikimo savybes.

Analgezija - viena iš probleminių medicinos sričių. Analgetikai (nuskausminamieji vaistai) naudojami siekiant apmalšinti paciento juntamą skausmą, taip pagerinant jo gyvenimo kokybę. Tačiau, naudojant analgetikus, susiduriama su nepageidaujamo poveikio paciento organizmui rizika.

Siekiant pagerinti žmonių, kuriems būtina pastovi analgezija, gyvenimo kokybė, siūloma išeitis - paciento kontroliuojama analgezija (PKA; angl. *Patient-Controlled Analgesia*). Tai yra vienas iš medicinoje naudojamų skausmo valdymo metodų.

1.1. Tikslai ir uždaviniai

Šio darbo tikslas - sukurti virtualų imitacinį paciento kontroliuojamos analgezijos modelį (PKA) ir ištirti paciento valdomos analgezijos vaistų pompos veikimo algoritmus (konvencinis, veikiantis boluso principu ir inovatyvus infuzinis) . Modelį sudaro kelios dalys: skausmo generatoriaus modelis, vaistų pompos modelis ir farmakokinetinis paciento organizmo modelis. Skausmo generatoriaus ir vaistų pompos modeliai, papildomai turi kelias variacijas skirtingiems pompos veikimo scenarijams patikrinti.

Pagrindinis uždavinys - atlikus algoritmų tyrimus, pagal gautuosius rezultatus, patvirtinti arba paneigti inovatyvaus algoritmo pranašumus, lyginant su įprastiniu algoritmu. Taip pat numatyti galimų tolimesnių tyrimų kryptis.

1.2. Dokumento paskirtis

Šiame dokumente aprašoma probleminė sritis, pateikiami galimi sprendimo metodai, jų realizacija ir tyrimas, siekiant nustatyti, kuris iš sprendimo metodų yra efektyvesnis.

2. PROBLEMINĖ SRITIS

2.1. Paciento kontroliuojama analgezija

Paciento kontroliuojama analgezija (PKA) yra bet kuris metodas, suteikiantis pacientui galimybę valdyti analgezijos vaistų dozavimą priklausomai nuo jo savijautos. Tokiu būdu pasiekiamas maksimalus įmanomas paciento komforto lygis. Tiesa, visiškos šio proceso kontrolės perleisti pacientui negalima. Todėl yra būtina, kad pacientas būtų stebimas kompetentingų asmenų, o naudojant medicininę įrangą, įrangos veikimo algoritmai su adekvačiais parametrų nustatymais, neleistų pacientui viršyti numatytos gaunamo vaisto per tam tikrą laiką dozės.[7] Esant visiems būtiniems priežiūros ir ribojimo mechanizmams, paciento kontroliuojama analgezija yra pranašesnė už tradicines opioidų injekcijas į raumenis.[1][6]

Šiame darbe tiriami medicininio prietaiso - paciento valdomos analgezijos vaistų pompos galimi veikimo algoritmai:

- įprastas, boluso principu paremtas algoritmas;
- inovatyvus, infuzijos principu paremtas algoritmas.

Paprastumo dėlei, toliau šiuos du algoritmus vadinsime atitinkamai PKA pompa ir IPKA pompa.

2.2. PKA pompa

Įprastinė PKA pompa pasižymi tuo, kad veikia boluso principu. Jos veikimo algoritmas yra labai paprastas:

- prieš naudojant, nustatomas vaistų dozės dydis ir pompos aktyvacijos trukmė;
- pacientas pareikalauja vaistų dozės;
- jei pompa neaktyvi - iškarto suleidžiama nustatytoji vaistų dozė;
- kitu atveju, vaistų dozės pareikalavimas ignoruojamas.

Vaistų dozė suleidžiama per sąlyginai trumpą laiką, lyginant su pompos aktyvacijos laiko trukme. Tai sukelia staigius vaistų koncentracijos svyravimus paciento kraujo plazmoje. Kadangi medicinoje naudojami analgeziniai medikamentai ilgu terapijos laikotarpiu pasižymi neigiamomis savybėmis paciento organizmui, kartu su minėtaisiais koncentracijos svyravimais gali iššaukti nepageidaujamus šalutinius poveikius bei pakenkti paciento sveikatai.

Taip pat, reikia atkreipti dėmesį ir į tą faktą, kad staigiai suleisti vaistai dažnu atveju viršija terapinę koncentraciją ir tuo pačiu būna greičiau pašalinami iš organizmo. Dėl šių priežasčių, vaistai naudojami neefektyviai, daromas neigiamas poveikis paciento sveikatai.[9]

2.3. IPKA pompa

IPKA pompos pranašumas prieš aptartąją PKA pompą - vaistas suleidžiamas tolygiu tempu, viso pompos veikimo metu.

Pompos algoritmas:

- prieš naudojant, nustatomas vaistų dozės dydis ir pompos aktyvacijos trukmė;
- pacientas pareikalauja naujos vaisto dozės;
- jei pompa neaktyvi, pradedamas vaisto dozės pristatymas;
- kitu atveju, nutraukiamas esamos dozės pristatymas ir pradedamas naujos dozės pristatymas.

Toks pompos veikimo algoritmas suteikia galimybę pasiekti norimą vaisto koncentracijos paciento kraujo plazmoje lygį su mažesnėmis svyravimų amplitudėmis. Taip sumažinama terapinio vaistų koncentracijos lygio viršijimo tikimybė, kadangi nepasiekus norimos vaisto koncentracijos ir pacientui pareikalavus naujos dozės, ankstesnės dozės pristatymas yra nutraukimas. Blogiausiu galimu atveju, kuomet pacientas gali pareikalauti naujos dozės prieš pat pasiekiant reikiamą koncentracijos lygį, bus viršytas koncentracijos lygis mažiau, negu įprastinės PKA pompos atveju, kadangi vaistas bus suleistas tolygiai.

2.4. Keliamos hipotezės

Pagal turimus pompos algoritmų aprašymus iš anksto galime daryti šias hipotezes, kurias tyrimo metu bandysime patvirtinti arba paneigti:

- PKA pompos atveju, kuo dažniau pacientas reikalaus naujos vaistų dozės, tuo mažesnė tikimybė, kad jis ją gaus;
- IPKA pompos atveju, kuo dažniau pacientas reikalaus naujos vaisto dozės, tuo ilgiau truks nepertraukiamas vaisto suleidimas;
- naudojant IPKA algoritmą, vaistų koncentracijos lygio kitimas yra tolygesnis negu PKA algoritmo atveju;
- staigaus vaisto suleidimo atveju, sudėtinga vaisto koncentraciją palaikyti norimame lygyje;
- blogiausio realaus galimo IPKA algoritmo bandymo scenarijaus koncentracijos pokytis didesnis negu PKA algoritmo atveju prie tų pačių vaisto dozės ir pompos aktyvios būsenos laiko parametrų.

- įmanomas scenarijus, kuomet, naudojant IPKA algoritmą, galima vaisto koncentraciją palaikyti artimą norimai terapinei koncentracijai

3. TYRIMAS

Siekiant nustatyti PKA ir IPKA algoritmų pranašumus ir trūkumus vienas kito atžvilgiu, būtina sudaryti imitacinį modelį, gebantį imituoti paciento elgseną, vaistų pompos veikimą ir vaistų koncentracijos kitimą paciento organizme.

3.1. Imitacinis modelis

Imitacinį modelį sudarysime iš kelių skirtingų mažesnių modelių**,** atliekančių tik jiems būdingas funkcijas. Šie modeliai bus sujungti į vieną bendrą sistemą.

Paciento elgsenos modelis:

Imituojamas paciento elgesys, tai yra, pagal iš anksto numatytus parametrus bei algoritmus, nustatomi laiko momentai, kuomet pacientas pareikalaus naujos vaisto dozės. Šiame darbe bus naudojami du paciento elgsenos algoritmai:

- atsitiktinis vaistų pareikalavimo pasiskirstymo algoritmas, kuomet generuojamas tam tikras kiekis atsitiktinių laiko momentų tam tikrai imitacijos trukmei;
- vaisto pareikalavimo algoritmas, priklausantis nuo skausmo lygio, kuomet vaisto pareikalavimo momentas priklauso nuo iš anksto numatytos vaisto koncentracijos paciento organizme.

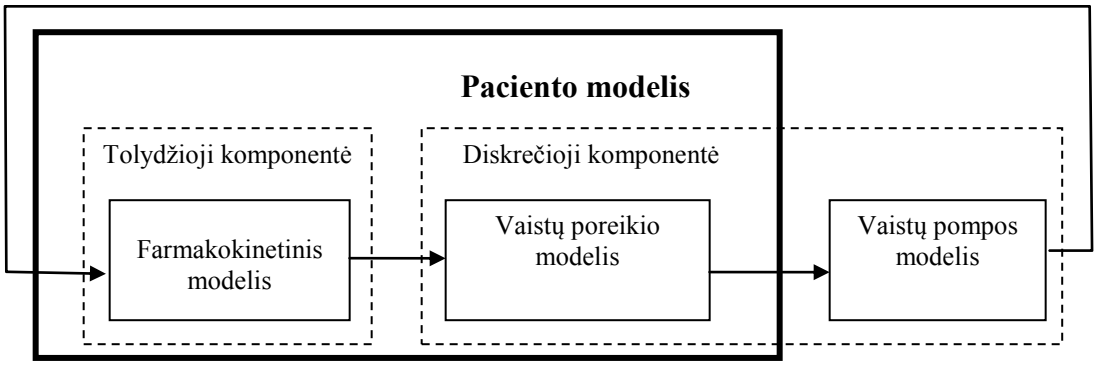
Farmakokinetinis paciento modelis:

Imituojamas vaisto pasiskirstymo paciento organizme procesas. Tai esminė viso imitacinio modelio dalis, leisianti nustatyti vaistų pompos algoritmų veikimo privalumus ir trūkumus.

Vaistų pompos modelis:

Pagal anksčiau aptartus PKA ir IPKA algoritmus, gavus užklausas iš paciento elgsenos modelio, generuoja laiko momentus, kuomet bus suleidžiami vaistai.

Apjungus išvardintuosius modelius į bendrą sistemą, gauname tokį PKA imitatoriaus modelį:



Pav. 1 PKA imitatoriaus modelio veikimo schema

3.1.1. Vaistų poreikio modelis

Vaisto pareikalavimo laiko momentams generuoti bus naudojami du skirtingi modeliai, realizuojantys, jau aptartus, skirtingus laiko reikšmių generavimo algoritmus. Modelius vadinsime atitinkamai skausmo generatorius A ir skausmo generatorius B.

3.1.1.1. Skausmo generatorius A

Skausmo generatorius A remiasi atsitiktinio vaistų pareikalavimo pasiskirstymo algoritmu:

- iš anksto nusprendžiama, kiek vidutiniškai kartų valandos bėgyje pacientas pareikalaus naujos vaistų dozės;
- skaičiuojame laiko intervalą iki sekančios vaistų dozės pareikalavimo.

Laiko intervalo skaičiavimui naudojame formulę:

$$\xi_j = -\frac{1}{\lambda} \cdot \ln \varsigma \quad (1), \text{ kai}$$

ξ_j – laiko intervalas tarp dviejų vaistų dozės pareikalavimų;

$j = 1,2,3...n$ - pareikalautos vaisto dozės numeris;

n – visų pareikalavimų skaičius;

λ – koeficientas, nurodantis vaisto dozės pareikalavimo dažnumą per valandą (pareikalavimo intensyvumas tiesiogiai proporcingas koeficiento dydžiui, t. y. koeficientui didėjant intensyvumas atitinkamai didėja);

ς –atsitiktinis skaičius intervale [0; 1].

3.1.1.2. Skausmo generatorius B

Skausmo generatorius B remiasi vaisto pareikalavimo algoritmu, priklausančiu nuo skausmo lygio:

- nusprendžiama, kokia yra minimali leistina vaistų koncentracija paciento organizme C_{min};
- duotuoju laiko momentu lyginama esama vaistų koncentracija C su numatytąja C_{min};
- jei $C < C_{min}$, tuomet skaičiuojamas laiko intervalas iki naujos vaisto dozės pareikalavimo. Laiko intervalo skaičiavimui naudojame formulę:

$$\xi_j = \varsigma(b-a)+a \quad (2), \text{ kai}$$

ξ_j – laiko intervalas tarp dviejų vaistų dozės pareikalavimų;

$j = 1,2,3...n$ - pareikalautos vaisto dozės numeris;

n – visų pareikalavimų skaičius;

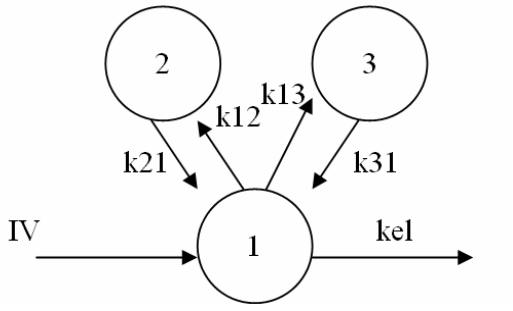
a - minimalus galimas laiko intervalas;

b - maksimalus galimas laiko intervalas;

ς –atsitiktinis skaičius intervale [a, b].

3.1.2. Farmakokinetinis paciento modelis

Farmakokinetinio modelio paskirtis - modeliuoti vaistų pasisavinimo, pasiskirstymo, metabolizmo ir pašalinimo iš organizmo procesus. Minėtiesiems procesams imituoti, naudojami vienos ar kelių sekcijų, vadinamų kompartamentais, modeliai. Kiekvienas kompartamentas atitinka tam tikrą žmogaus kūno dalį - audinių grupę organą, organų sistemą. Šiame darbe bus naudojamas trijų kompartamentų farmakokinetinis modelis. Pav. 2 vaizduojama trijų kompartamentų modelio schema. Rodyklės , jungiančios kompartamentus, rodo vaistų judėjimo kryptį paciento organizme. Koeficientai prie rodyklių rodo srauto judėjimo stiprumą (kokia vaistų dalis pereina į kitą kompartamentą per laiko vienetą). [6]



Pav. 2 Trijų kompartamentų farmakokinetinis modelis

Mūsų naudojamame modelyje laikysime, kad suleidus vaistus į pirmąjį kompartamentą, vaisų koncentracija jame padidėja šuoliškai.

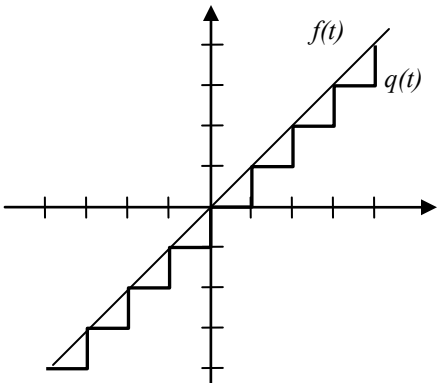
Modelis aprašomas diferencinėmis lygtimis:

$$\begin{cases} \frac{dX_1}{dt} = k21 \cdot X_2 - k12 \cdot X_1 + k31 \cdot X_3 - k13 \cdot X_1 - kel \cdot X_1 \\ \frac{dX_2}{dt} = k12 \cdot X_1 - k21 \cdot X_2 \\ \frac{dX_3}{dt} = k13 \cdot X_1 - k31 \cdot X_3 \end{cases}, (3)$$

X_1, X_2 ir X_3 yra atitinkamų kompartamentų vaistų koncentracija kraujo plazmoje. Vaistų koncentracijos kitimas priklauso nuo vaistų judėjimo tarp kompartamentų intensyvumo ir vaistų šalinimo iš organizmo, priklausančio nuo konstantos *kel*. Vaistų suleidimas *IV* lygčių sistemoje nėra žymimas, nes suleidimo metu skaičiuojant funkcijos reikšmę nenaudojamos diferencialinės lygtys, o tiesiog padidinama funkcijos reikšmė nustatytu dydžiu:

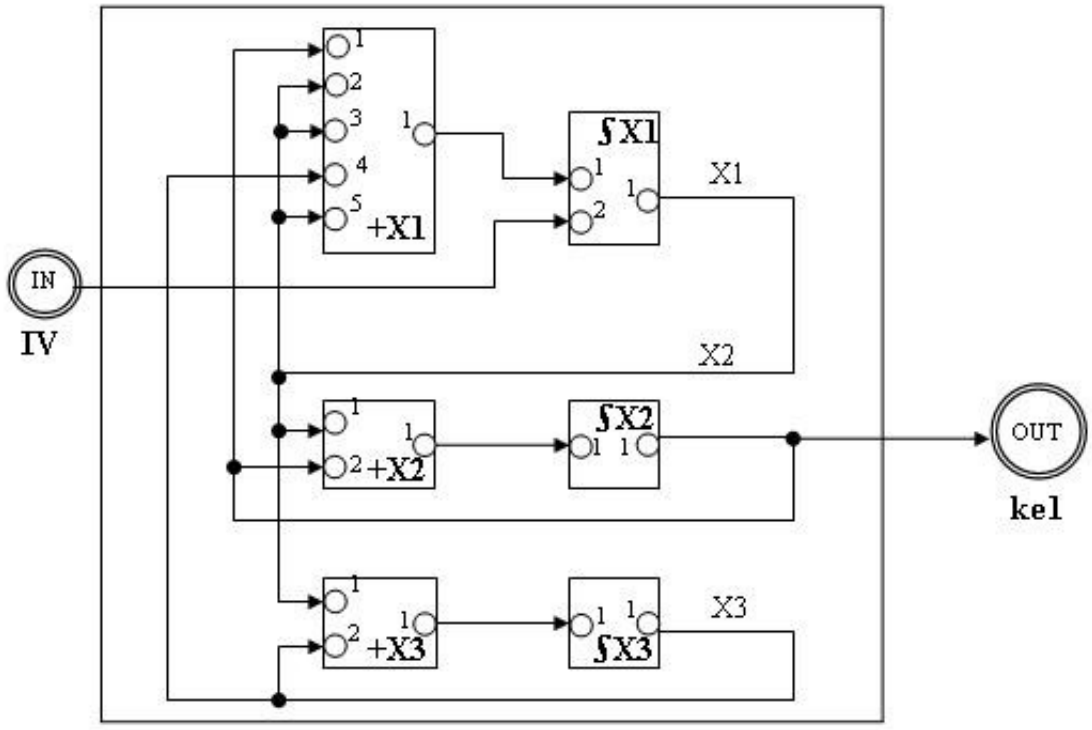
$$X_1' = X_1 + IV \text{ .(4)}$$

Farmakokinetiniai skaičiavimai atliekami juos diskretizuojant ne laiko atžvilgiu, kaip yra įprasta,o vaistų koncentracijos funkcijos reikšmės pokyčio žingsnio atžvilgiu. Tokiu būdu gauname diskrečią funkcijos reikšmių seką. Diskretizavimui įgyvendinti, pasinaudosime QSS (*Quantized State System,* liet. *kvantuotos būsenos sistema)* modeliu.[2] QSS modelis sudarytas iš dviejų pagrindinių elementų: funkcijos reikšmę skaičiuojančio integratoriaus, ir funkcijos reikšmės kvantavimo funkcijos. QSS modelio integratorių galima apbūdinti kaip objektą, kuris priima skaičiuojamos funkcijos diferencialą ir skaičiuoja funkcijos reikšmę apibrėžtame reikšmių tinklelyje. Objekto būsena priklauso nuo pradinės jo būsenos ir įeinančios diferencialo reikšmės. Jei išorinis signalas nesikeičia, sistema veikia kaip normali diferencialinių lygčių sistema. Pasikeitus įėjimo signalui, diferencialo reikšmę aprašanti išraiška irgi kinta.



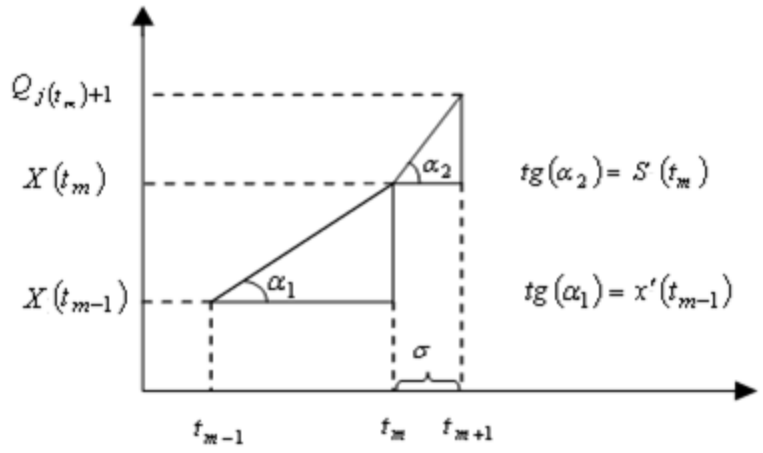
Pav. 3 Diskretizavimo funkcija

Tokiu atveju, farmakokinetinį modelį galime atvaizduoti, kaip agregatą, sudarytą iš smulkesnių agregatų - integratorių ir sumatorių:



Pav. 4 Agregatinė farmakokinetinio modelio schema

Pav. 4 vaizduojamoje modelio schemoje +X1, +X2, +X3 - sumatoriai, atitinkantys dešiniąją (3) lygčių sistemos dalį. ∫X1, ∫X2, ∫X3 - integratoriai, atitinkantys kairiąją (3) lygčių sistemos dalį. Kadangi integratoriai veikiami išorinių ir vidinių įvykių, įvykių laiko tarpas σ, per kurį funkcija X pasieks sekančią kvantuotą reikšmę po įvykio, skaičiuojamas taip:



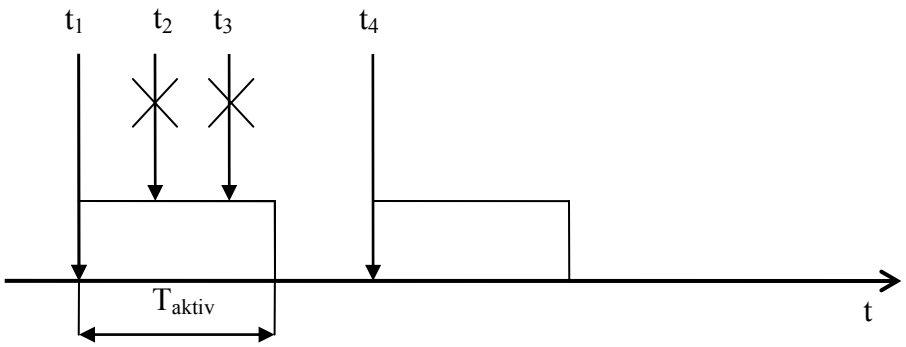
Pav. 5 Pagalbinis grafikas σ skaičiavimui

Čia $X(t_m) \in R$ - apskaičiuota funkcijos X reikšmė, $X(t_{m-1})$ - atitinkama reikšmė vienu elementariu laiko intervalu prieš tai, $Q_j(t_m)+1$ - funkcijos sekanti kvantuota reikšmė, $S_1(t_m) \in R$ – skaičiuojamos funkcijos išvestinė. Nuo pastarosios reikšmės priklauso ar funkcijos reikšmės keisis, pasiekus sekantį laiko momentą: didės ($S(t_m) > 0$), mažės ($S'(t_m) < 0$) ar nesikeis ($S(t_m) = 0$). $x'(t_{m-1}) \in R$ yra funkcijos išvestinė vienu elementariu laiko intervalu anksčiau.

3.1.3. Vaistų pompos modelis

Vaistų pompos modelis imituoja vaistų suleidimo proceso valdymą. T.y. atitinkamai pagal algoritmą, po paciento naujos vaistų dozės pareikalavimo, priklausomai nuo pompos aktyvumo būsenos, skaičiuoja naują laiko momento reikšmę, kuomet pompa pereis į neaktyvią būseną.

3.1.3.1. PKA algoritmo modelis

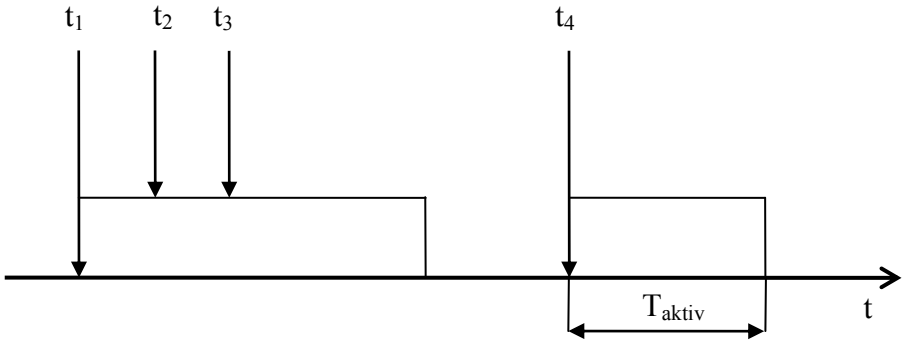


Pav. 6 Principinė PKA pompos modelio darbo grafikas

Pagal pateiktą grafiką, matome, kad pacientui pareikalavus naujos dozės laiko momentu t_1, pompa perėjo į aktyvią būseną, o anksčiausias laiko momentas, kuomet pompa vėl bus neaktyvi, $T = t_1 + T_{aktiv}$. Pacientui paprašius naujų dozių laiko momentais t_2 it t_3, pompa į prašymus nereagavo, išliko aktyvioje būsenoje, o laikas T nepakito. Aktyvios būsenos metu nauji prašymai atmetami siekiant išvengti galimo vaistų perdozavimo, kadangi vaistų dozė suleidžiama per sąlyginai trumpą momentinį laiką.

Laiko momentu T pompa vėl perėjo į neaktyvią būseną, kas leido sėkmingai paprašyti naujos vaistų dozės laiko momentu t_4.

3.1.3.2. IPKA algoritmo modelis



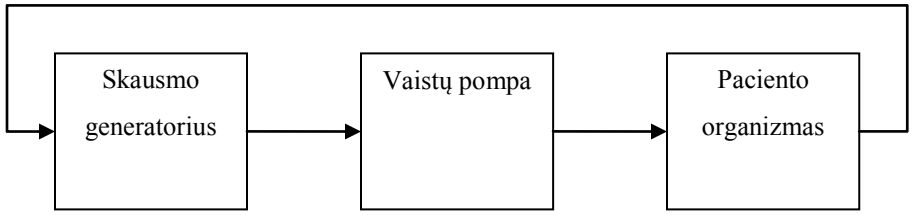
Pav. 7 Principinė IPKA pompos modelio darbo grafikas

Taip pat, kaip ir PKA pompa, IPKA pompa, gavusi pakartotinius užklausimus t ir t, toliau išlieka aktyvioje būsenoje, tačiau kiekvieno tokio užklausimo atveju skaičiuojamas naujas laikas $T = t_{užklausimo} + T_{aktiv}$. Į neaktyvią būseną pompa pereina tik laiko momentu T.

Naujos užklausos aktyvios būsenos metu nėra atmetamos, kadangi vaistai yra leidžiami visos aktyvios būsenos metu pastoviu tempu.

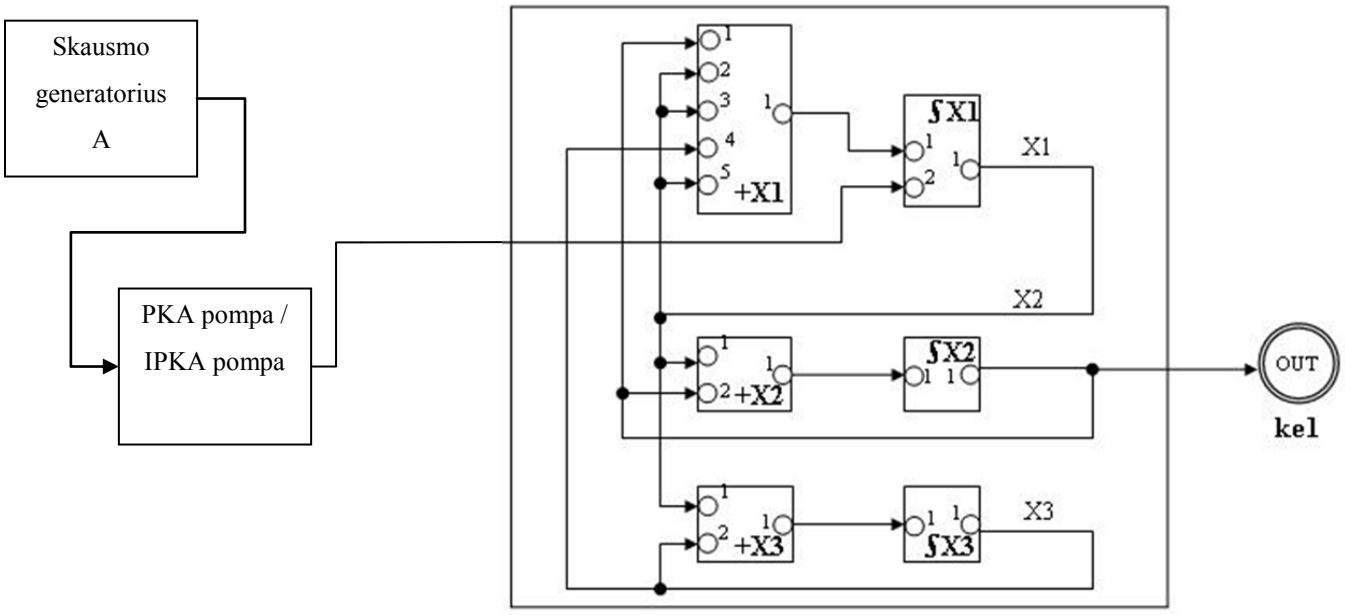
3.1.4. Agregatinis PKA modelis

Verta atkreipti dėmesį, kad imitacinis PKA modelis (pav. 8) turi hibridinėms sistemoms būtinas savybes: sudarytas iš diskrečiųjų ir tolydžiųjų komponentų. [3][8] Todėl galima imitacinio PKA modelio komponentus atvaizduoti kaip agregatus:

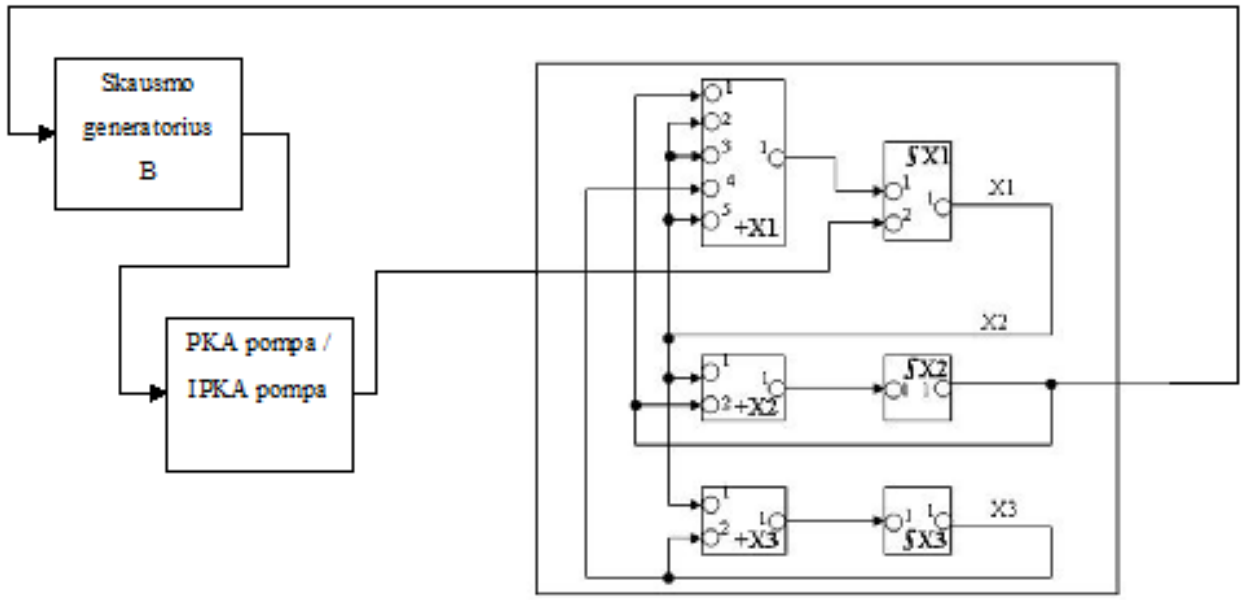


Pav. 8 Bendras agregatinis imitacinis PKA modelis

Pateikiame ir detalizuotas agregatinio imitacinio modelio variacijų schemas:



Pav. 9 Agregatinė imitacinio PKA modelio schema, kai naudojamas skausmo generatorius A



Pav. 10 Agregatinė imitacinio PKA modelio schema, kai naudojamas skausmo generatorius B

3.2. Įvadas į PLA

Siekiant išvengti įvairių problemų, kylančių sistemos projektavimo metu, PKA imitacinio modelio agregatų vienareikšmiškam nusakymui buvo panaudota virtualių modelių specifikacija agregatiniu metodu. Agregatinio sistemos specifikavimo požiūriu, sistema suprantama, kaip tarpusavyje sąveikaujančių atkarpomis tiesinių agregatų (angl. *Piece-Linear Aggregates - PLA*) aibė A. [4]

Kiekvienam agregatui $A_i \in A$ turi būti apibrėžta:

1. Įėjimo signalų aibė X_i.
2. Išėjimo signalų aibė Y_i.
3. Išorinių įvykių aibė E'_i, kurią sudaro išoriniai įvykiai $e'(x)$ susieti su įėjimo signalais $x \in X_i$.
4. Vidinių įvykių aibė E''_i.
5. Valdymo sekos, kurios apibūdina agregato A_i vidinių įvykių trukmes. Jos gali būti išreikštos tiesiogine laiko trukme arba intensyvumu.
6. Diskrečioji agregato A_i būsenos dedamoji $v_i(t)$.

7. Tolydžioji agregato A_i būsenos dedamoji $z_{vi}(t) = \bigcup_{j=1}^{r} w(e“_j, t)$, kur $w(e“_j, t)$ yra tolydusis kintamasis, susietas su vidiniu įvykiu $e“_j \in E“_i$, o r – vidinių įvykių skaičius aibėje $E“_i$.
8. Agregato būseną $z_i(t)$ sudaro diskrečioji komponentė $v_i(t)$ ir tolydžioji komponentė $z_{vi}(t)$: $z_i(t) = (v_i(t), z_{vi}(t))$.
9. Kiekvienas vidinis ir išorinis įvykis turi du operatorius: H ir G. Operatorius H (perėjimo arba atvaizdavimo operatorius) keičia diskrečiųjų ir tolydžiųjų agregato kintamųjų reikšmes, o G (išėjimo operatorius) – formuoja išėjimo signalus Y_i.

Imitacinio modelio agregatų specifikacijose laikui žymėti naudojama: t_0 – pradinis simuliuojamas laiko momentas, t_m – esamas simuliuojamo laiko momentas ir t_{m+i} – i laiko momentų vėlesnis simuliuojamo laiko momentas.

3.3. Formalus agregatų aprašymas

3.3.1. Skausmo generatorius A

1. X = Ø
2. Y = {y1}; $y_1 = 1$ – sugeneruotu vaisto pareikalavimo simuliuojamu momentu grąžinamas loginis vienetas
3. E’ = Ø
4. E’’ = {e_1’’}
5. E’’ = {ξ_j}; $\xi_j = -\frac{1}{\lambda} \cdot \ln \varsigma$, kur $\varsigma \in [0;1]$ ir yra atsitiktinis generuojamas dydis, o λ – vidutinis vaisto pareikalavimo intensyvumas per valandą
6. $V(t_m) = Ø$
7. $Z_v(t_m) = w(e_1’’, t_m)$
8. $w(e_1’’, t_0) = \xi_t$
9. H(e_1’’):

   $w(e_1’’, t_{m+1}) = t_m + \xi_j$ …
10. G(e_1’’): y_1.

3.3.2. Skausmo generatorius B

1. X = {C} C - vaisto koncentracija pasirinktajame kompartamente
2. Y = {y1}; $y_1 = 1$ – sugeneruotu vaisto pareikalavimo simuliuojamu momentu grąžinamas loginis vienetas
3. E'= {e_1'}
4. E" = {e_2"}
5. e_1" → { ξ_j }
6. $V(t_m) = C_{min}$, C_{min} - minimali vaisto koncentracija pasirinktajame kompartamente
7. $Z_v(t_m) = w(e_1'', t_m) = t_m + \xi$
8. $V(t_0) = C_{min}$

   $w(e_1'', t_0) = \infty$
9. $H(e_1')$:

   $w(e_1'', t_{m+1}) = t_m + \xi_j$, jei $C < C_{min}$

   $w(e_1'', t_m)$, jei $C \geq C_{min}$

   $H(e_1'')$:

   $w(e_1'', t_{m+1}) = \infty$
10. $G(e_1'')$: y = 1.

3.3.3. PKA pompa

1. X = {x_1}, x_1 – pareikalauta vaisto dozės
2. Y = {y_1}, y_1 – analizės rezultatai
3. E'= {e_1'}
4. E'' = {e_1'', e_2''},

   e_1'' – laikas dozės pristatymui baigėsi;

   e_2'' – simuliacijos laikas baigėsi
5. $e_1'' \rightarrow \{T\}$, T – parametras; T = 10 min. numatytuoju atveju; T - laikas, skirtas pristatyti dozę

   $e_2'' \rightarrow \{T_{sim}\}$; T_{sim} – simuliuojamo terapijos laiko limitas
6. $V(t_m) = \{q(_{tm}), n_r(t_m), n_d(t_m), t_1(t_m)\}$

   $q(t_m) =$ { 0; būsena, kai pompa laukia aktyvacijos; 1; būsena, kai pompa užblokuota

   $n_r(t_m)$ – pareikalautų dozių skaičius

   $n_d(t_m)$ – pristatytų dozių skaičius

   $t_1(t_m)$ – laiko momentas, kai pompa bus atblokuota

7. $Z_v(t_m) = \{w(e_1'', t_m), w(e_2'', t_m)\}$
8. $q(t_m) = 0$; $n_r(t_m) = 0$; $n_d(t_m) = 0$; $t_1(t_m) = 0$.

9. $H(e_1')$:

$n_r(t_{m+1}) = n_r(t_m) + 1$

$q(t_{m+1}) = 1$

$w(e_1'', t_{m+1}) = t_m + T$, **jei** $q(t_m) = 0$

$w(e_1'', t_{m+1}) = w(e_1'', t_m)$, **jei** $q(t_m) = 1$

$t_{wm} = t_m - t_1$, **jei** $q(t_m) = 0$; t_{wm} – dozės laukimo vidurkis

$stt(t_{wm})$;

$H(e_1'')$:

$n_d(t_{m+1}) = n_d(t_m) + 1$

$q(t_{m+1}) = 0$

$w(e_1'', t_{m+1}) = \infty$

$t_1 = t_m$

$H(e_2'')$:

$$P_d = \frac{n_d(t_m)}{n_r(t_m)};\ P_{nd} = 1 - P_d\ .\ P_d$$ – dozės pristatymo tikimybė, P_{nd} – atvirkščias P_d dydis

10. $G(e_2'')$: y_1.

3.3.4. IPKA pompa

1. $X = \{x_1\}$, x_1 – pareikalauta dozės
2. $Y = \{y_1\}$, y_1 – analizės rezultatai
3. $E' = \{e_1'\}$
4. $E'' = \{e_1'', e_2''\}$,

   e_1'' – baigėsi vaistų leidimas;

   e_2'' – baigėsi simuliacijos laikas
5. $e_1'' \rightarrow \{T\}$, valdymo seką formuoja signalo atėjimas

   $e_2'' \rightarrow \{T_{sim}\}$; T_{sim} – simuliacijos laiko limitas.
6. $V(t_m) = \{q(_{tm}), n_r(t_m), n_d(t_m), t_1(t_m)\}$

$q(t_m)$ = { 0; būsena, kai pompa laukia aktyvacijos
1; būsena, kai pompa veikia

$n_r(t_m)$ – pareikalautų dozių skaičius

$n_d(t_m)$ – pristatytų dozių skaičius

$t_1(t_m)$ – laiko momentas, kai pompa pasieks neaktyvią būseną

$t_2(t_m)$ – laiko momentas, kai vaistai bus pradėti pristatyti

7. $Z_v(t_m) = \{w(e_1'', t_m), w(e_2'', t_m)\}$

8. $q(t_m) = 0$; $n_r(t_m) = 0$; $n_d(t_m) = 0$; $t_1(t_m) = 0$; $t2(tm) = -\infty$

9. $H(e_1')$:

$n_r(t_{m+1}) = n_r(t_m) + 1$

$q(t_{m+1}) = 1$

$w(e_1'', t_{m+1}) = t_m + T$

$t_{wm} = t_m - t_1$, if $q(t_m) = 0$; t_{wm} – vidurkis

$stt(t_{wm})$;

$t_2(t_{m+1}) = t_m$, if $q(t_m) = 0$

$H(e_1'')$:

$n_d(t_{m+1}) = n_d(t_m) + 1$

$t_1(t_{m+1}) = t_m$

$t_{dm} = t_m - t_2(t_m)$; t_{dm} – vaistų pristatymo intervalo vidurkis

$stt(t_{dm})$

$H(e_2'')$:

Output: $n_r(t_{m+1})$, $n_d(t_{m+1})$, $M(t_{wm})$, $M(t_{dm})$

10. $G(e_2'')$: y_1.

Suvartotų vaistų kiekis lygus $d = n_d(T_{sim}) \cdot v$, kur d – vaistų kiekis, o v – infuzijos tempas.

3.3.5. Integratorius

1. Įėjimo signalų aibė $X = \{S_1(t_m), x_y\}$,

čia: $S_1(t_m) \in R$ - skaičiuojamos funkcijos išvestinė ($S_1(t_m) = \frac{dX_1}{dt}$ iš +X1),

$x_y \in R$ - momentinio funkcijos pokyčio reikšmė (iš generatoriaus) – t.y., momentinis vaistų suleidimas.

2. Išėjimo signalų aibė $Y = Q_{j_1}(t_m), j = 1 \ldots r$ - funkcijos X_1 (vaistų konc. pirmojoje sekcijoje) kvantuota reikšmė,
3. Išorinių įvykių aibė $E' = \{e_1', e_2'\}$,

   čia: e_1' - atėjo pasikeitusi išvestinės reikšmė,

   e_2' - atėjo signalas, keičiantis funkcijos X_1 (vaistų konc. pirmojoje sekcijoje), reikšmę,
4. Vidinių įvykių aibė $E'' = \{e_1''\}$,

   čia: e_1'' - funkcija X_1 (vaistų konc. pirmojoje sekcijoje) pasiekė kitą kvantinį slenkstį,
5. Diskrečioji agregato būsenos dedamoji $v(t_m) = \{X_1(t_m), x_1'(t_m), j_1(t_m)\}$

   čia $X_1(t_m) \in R$ - apskaičiuota funkcijos X_1 reikšmė,

   $x_1'(t_m) \in R$ - esama funkcijos išvestinės reikšmė,

   $j_1(t_m) \in Z$ - funkcijos X_1 kvantuotos būsenos numeris,
6. Tolydžioji būsenos dedamoji $z_v(t_m) = \{w(e_1'', t_m)\}$ - laiko momentas, kada funkcija X_1 pasieks naują kvantinį slenkstį,

$$w(e_2'', t_m) = \begin{cases} < \infty, x_1'(t_m) \neq 0 \\ \infty \ priešingu \ atveju \end{cases}$$

7. Valdymo sekos $e_1' \mapsto \{\sigma_1\}$, $e_1'' \mapsto \{\sigma_2\}$, $e_2' \mapsto \{\sigma_3\}$

   čia: σ_1 yra laiko tarpas, per kurį funkcija X_1 pasieks sekančią kvantuotą reikšmę po išorinio įvykio. σ_2 – laiko tarpas, per kurį funkcija X_1 pasieks sekančią kvantuotą reikšmę po vidinio įvykio, σ_3 – laiko tarpas, po funkcijos X_1 padidinimo (dydžiu y_r):

8. $$\sigma_1 = \begin{cases} \dfrac{Q_{j_1(t_{m-1})+1} - \left(X_1(t_{m-1}) + (t_m - t_{m-1}) \cdot x_1'(t_{m-1})\right)}{S_1(t_m)} & kai \ S_1(t_m) > 0 \\ \dfrac{\left(X_1(t_{m-1}) + (t_m - t_{m-1}) \cdot x_1'(t_{m-1})\right) - \left(Q_{j_1(t_{m-1})-1}\right)}{|S_1(t_m)|} & kai \ S_1(t_m) < 0 \\ \infty \quad kai \ S_1(t_m) = 0 \end{cases}$$

$$\sigma_2 = \begin{cases} \dfrac{Q_{j_1(t_{m-1})+2} - \left(X_1(t_{m-1}) + (t_m - t_{m-1}) \cdot x_1'(t_{m-1})\right)}{x_1'(t_{m-1})} & kai\ x_1'(t_{m-1}) > 0 \\ \dfrac{\left(X_1(t_{m-1}) + (t_m - t_{m-1}) \cdot x_1'(t_m)\right) - \left(Q_{j_1(t_{m-1})-1}\right)}{\left|x_1'(t_{m-1})\right|} & kai\ x_1'(t_{m-1}) < 0 \\ \infty \quad kai\ x_1'(t_m) = 0 \end{cases}$$

ir

$$\sigma_3 = \begin{cases} \dfrac{Q_{j_1(t_{m-1})+\lfloor (t_m - t_{m-1}) \cdot x_1'(t_{m-1}) + x_y / \Delta Q \rfloor + 1} - \left(X_1(t_{m-1}) + (t_m - t_{m-1}) \cdot x_1'(t_{m-1}) + x_y\right)}{x_1'(t_{m-1})} & kai\ x_1'(t_{m-1}) > 0 \\ \dfrac{\left(X_1(t_{m-1}) + (t_m - t_{m-1}) \cdot x_1'(t_{m-1}) + x_y\right) - \left(Q_{j_1(t_{m-1})+\lfloor (t_m - t_{m-1}) \cdot x_1'(t_{m-1}) + x_y / \Delta Q \rfloor}\right)}{\left|x_1'(t_{m-1})\right|} & kai\ x_1'(t_{m-1}) < 0 \\ \infty \quad kai\ x_1'(t_{m-1}) = 0 \end{cases}$$

Pastaba: Operatorius $\lfloor z \rfloor$ – skaičiaus *z* sveikoji dalis, pvz. $\lfloor 6.123 \rfloor = 6$.

9. Pradinė būsena:

$$v(t_0) = \{X_1(t_0), x_1'(t_0), j_1(f(X_1(t_0)))\}$$

$$z_v(t_0) = \{t_0 + \sigma_2\}$$

10. Perėjimo ir išėjimo operatoriai:

$H(e_1'(S_1(t_m)))$: / atėjo nauja išvestinės reikšmė $S_1(t_m)$ /

$X_1(t_m) = X_1(t_{m-1}) + (t_m - t_{m-1}) \cdot x_1'(t_{m-1})$

$x_1'(t_m) = S_1(t_m)$

$j_1(t_m) = j_1(t_{m-1})$

$w(e_1', t_{m+1}) = t_m + \sigma_1$

$H(e_2'(x_y))$: / atėjo funkcijos X_1 pokyčio reikšmė x_y /

$X_1(t_m) = X_1(t_{m-1}) + (t_m - t_{m-1}) \cdot x_1'(t_{m-1}) + x_y$

$x_1'(t_m) = x_1'(t_{m-1})$

$j_1(t_m) = j_1(t_{m-1}) + \lfloor (t_m - t_{m-1}) \cdot x_1'(t_{m-1}) + x_y / \Delta Q \rfloor$

$w(e_1', t_{m+1}) = t_m + \sigma_3$

$G(e_2')$: $y = Q_{j_1(t_m)+\lfloor (t_{m+1}-t_m)\cdot x_1'(t_m)+x_y/\Delta Q \rfloor}$

$H(e_1'')$:

$X_1(t_m) = X_1(t_{m-1}) + (t_m - t_{m-1}) \cdot x_1'(t_{m-1})$ / pasiekta nauja funkcijos X_1 kvantuota reikšmė /

$x_1'(t_m) = x_1'(t_{m-1})$

$j_1(t_m) = j_1(t_{m-1}) + \mathrm{sgn}(x_1'(t_{m-1}))$

$w(e_1'', t_{m+1}) = t_m + \sigma_2$

$G(e_1'')$: $y = Q_{j_1(t_{m-1})+\mathrm{sgn}(x_1'(t_{m-1}))}$

3.3.6. Sumatorius X1

+X1 – pirmojo kompartamento sumatorius

1. Įėjimo signalų aibė $X = \{X_i\}$ – skaičiuojamos sumos dėmenys,

   čia $X_i \in R, i \in \{1,2,3\}$– vaistų koncentracija *i*-ojoje sekcijoje (iš **∫X1 , ∫X2, ∫X3**) ,

2. Išėjimo signalų aibė $Y = S_1(t_m)$ – suskaičiuota suma (t.y., išvestinės reikšmė $S_1(t_m) = \frac{dX_1}{dt}$) ,

3. Išorinių įvykių aibė $E' = \{e_1'\}$,

   čia: e_1 – pasikeitęs sumos operandas (t.y., pasikeitė išvestinės reikšmė),

4. Vidinių įvykių aibė $E'' = \varnothing$;

5. Diskrečioji agregato būsenos dedamoji

   $v(t_m) = \{X_1(t_m), X_2(t_m), X_3(t_m), \beta_1 = -k12 - k13 - kel, \beta_2 = k21, \beta_3 = k31, S_1(t_m)\}$,

   čia $X_i(t_m) \in R, \quad i = \{1,2,3\}$ – sumos operandai (t.y. – vaistų koncentracija *i*-ojoje sekcijoje),

   $\beta_i(t_m) \in R, \quad i = \{1,2,3\}$ – iš anksto nustatyti sumos operandų koeficientai,

   $S_1(t_m)$– suskaičiuota išvestinės reikšmė,

6. Tolydžioji būsenos dedamoji $z_v(t_m)=\infty$

7. Valdymo sekos – $\varnothing$,

8. Pradinė

   būsena:

   $v(t_m)=\{X_1(t_0),X_2(t_0),X_3(t_0),\beta_1,\beta_2,\beta_3,S_1(t_0)\}=\{0,0,0,-k12-kel,k21,k31,0\}$,

   $z_v(t_0)=\infty$,

9. Perėjimo ir išėjimo operatoriai:

   $H(e_1'(x_k))$:

   $X_k(t_m)=x_k$

   $X_i(t_m)=X_i(t_{m-1}),\quad 1\le i\le 3,\quad i\ne k$

   $S_1(t_m)=X_1^*\cdot\beta_1+X_2^*\cdot\beta_2+X_3^*\cdot\beta_3$

   čia $X_i^*=\begin{cases}X_i(t_{m-1}), & i\ne k\\ x_k, & i=k\end{cases}\quad i=1,2,3$

   $z_v(t_m)=\infty$

10. $G(e_1'')$: $Y=X_1^*\cdot\beta_1+X_2^*\cdot\beta_2+X_3^*\cdot\beta_3$

3.3.7. Sumatorius X2

+X2 – antrojo kompartamentų sumatorius

1. Įėjimo signalų aibė $X=\{X_i\}$ – skaičiuojamos sumos dėmenys,

   čia $X_i\in R, i\in\{1,2\}$– vaistų koncentracija *i*-ojoje sekcijoje (iš **∫X1, ∫X2**),

2. Išėjimo signalų aibė $Y=S_2(t_m)$ – suskaičiuota suma (t.y., išvestinės reikšmė $S_2(t_m)=\frac{dX_2}{dt}$) ,

3. Išorinių įvykių aibė $E'=\{e_1'\}$,

   čia: e_1 – pasikeitęs sumos operandas (t.y., pasikeitė išvestinės reikšmė),

4. Vidinių įvykių aibė $E''=\varnothing$;

5. Diskrečioji agregato būsenos dedamoji

$v(t_m)=\{X_1(t_m), X_2(t_m), \beta_1 = k12, \beta_2 = -k21, S_2(t_m)\}$,

čia $X_i(t_m)\in R, \quad i=\{1,2\}$ – sumos operandai (t.y. – vaistų koncentracija i-ojoje sekcijoje),

$\beta_i(t_m)\in R, \quad i=\{1,2\}$ – iš anksto nustatyti sumos operandų koeficientai,

$S_2(t_m)$– suskaičiuota suma,

6. Tolydžioji būsenos dedamoji $z_v(t_m)=\infty$

7. Valdymo sekos – $\varnothing$,

8. Pradinė būsena: $v(t_m)=\{X_1(t_0), X_2(t_0), \beta_1, \beta_2, S_2(t_0)\}=\{0,0,k12,-k21,0\}$,

$z_v(t_0)=\infty$,

9. Perėjimo ir išėjimo operatoriai:

$H(e_1'(x_k))$:

$X_k(t_m)=x_k$

$X_i(t_m)=X_i(t_{m-1}), \quad 1\le i\le 2, \quad i\ne k$

$S_2(t_m)=X_1^*\cdot\beta_1+X_2^*\cdot\beta_2$

$\text{čia } X_i^*=\begin{cases} X_i(t_{m-1}), & i\ne k \\ x_k, & i=k \end{cases} \quad i=1,2$

$z_v(t_{m+1})=\infty$

10. $G(e_1'')$: $Y=X_1^*\cdot\beta_1+X_2^*\cdot\beta_2$

3.3.8. Sumatorius X3

+X3 – antrojo kompartamentų sumatorius

1. Įėjimo signalų aibė $X=\{X_i\}$ – skaičiuojamos sumos dėmenys,

čia $X_i\in R, i\in\{1,3\}$– vaistų koncentracija *i*-ojoje sekcijoje (iš ∫X1, ∫X3) ,

2. Išėjimo signalų aibė $Y=S_3(t_m)$ – suskaičiuota suma (t.y., išvestinės reikšmė $S_3(t_m)=\frac{dX_3}{dt}$) ,

3. Išorinių įvykių aibė $E' = \{e_1'\}$,

   čia: e_1 – pasikeitęs sumos operandas (t.y., pasikeitė diferencialo reikšmė),

4. Vidinių įvykių aibė $E'' = \varnothing$;

5. Diskrečioji agregato būsenos dedamoji

   $v(t_m) = \{X_1(t_m), X_3(t_m), \beta_1 = k13, \beta_3 = -k31, S_3(t_m)\}$,

   čia $X_i(t_m) \in R, \quad i = \{1,3\}$ – sumos operandai (t.y. – vaistų koncentracija i-ojoje sekcijoje),

   $\beta_i(t_m) \in R, \quad i = \{1,3\}$ – iš anksto nustatyti sumos operandų koeficientai,

   $S_3(t_m)$ – suskaičiuota suma,

6. Tolydžioji būsenos dedamoji $z_v(t_m) = \infty$

7. Valdymo sekos – $\varnothing$,

8. Pradinė būsena: $v(t_m) = \{X_1(t_0), X_3(t_0), \beta_1, \beta_3, S_3(t_0)\} = \{0,0, k13, -k31, 0\}$,

   $z_v(t_0) = \infty$,

9. Perėjimo ir išėjimo operatoriai:

   $H(e_1'(x_k))$:

   $X_k(t_m) = x_k$

   $X_i(t_m) = X_i(t_{m-1}), \quad i = 1,3, \quad i \neq k$

   $S_3(t_m) = X_1^* \cdot \beta_1 + X_3^* \cdot \beta_3$

   $\text{čia } X_i^* = \begin{cases} X_i(t_{m-1}), & i \neq k \\ x_k, & i = k \end{cases} \quad i = 1,3$

   $z_v(t_{m+1}) = \infty$

10. $G(e_1'')$: $Y = X_1^* \cdot \beta_1 + X_3^* \cdot \beta_3$

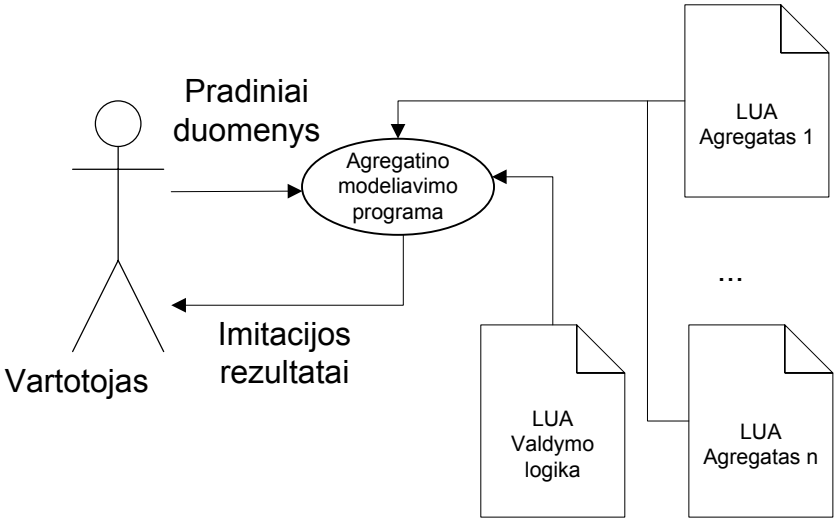
3.4. Modelio realizacijos aplinka ir priemonės

PKA modeliui realizuoti, buvo sukurta agregatinio modeliavimo programa (autoriai: Virginijus Pampikas ir Donatas Urbas), suteikianti galimybę norimus realizuoti modelio

agregatus aprašyti LUA skriptų programavimo kalba. Agregatų valdymo logika taip pat realizuojama naudojant LUA kalbą.

Tokios realizacijos aplinkos ir priemonių pasirinkimas suteikia galimybę lengvai ir greitai realizuoti reikiamus modelio elementus. Kadangi modelio elementai yra aprašyti PLA formalizme, vietos dviprasmybėms ir netinkamoms specifikacijos interpretacijoms nelieka. Todėl, iš esmės, realizavimo aplinkos ir priemonių pasirinkimui didžiausią įtaką daro realizuojančio asmens kompetencija programavimo ir įvairių programų paketų naudojimo srityje.

Šiame darbe naudotoji agregatinio modeliavimo programa suteikia plačias galimybes norimų agregatų kūrimui, kadangi kiekvienam agregatui aprašyti, yra privalomos realizuoti vos kelios sąsajos funkcijos, paliekant tolimesnę laisvę programuotojui pagal jo poreikius.



Pav. 11 Principinė naudojamos sistemos veikimo schema

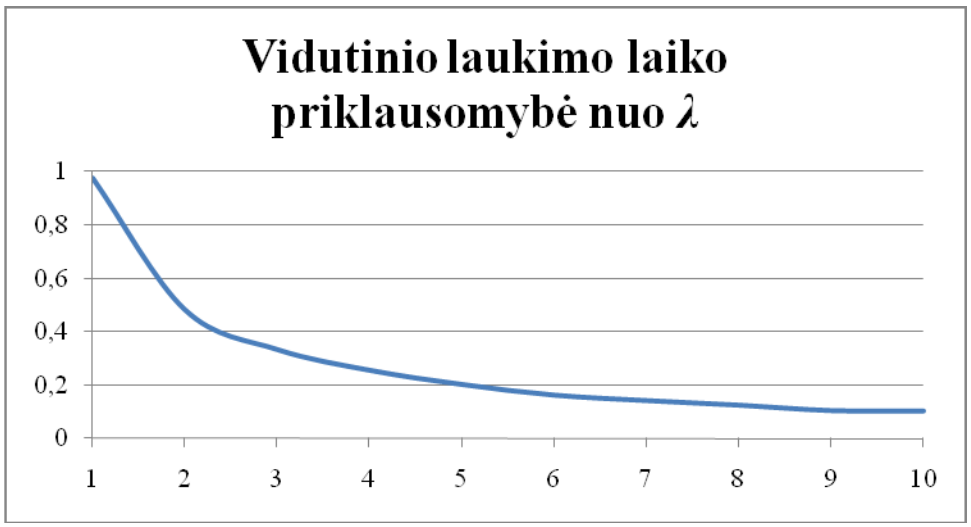
Realizuotų agregatų bei jų valdymo logikos išeities tekstai pridedami priede.

3.5. Modelio testavimas

Siekiant užtikrinti tinkamus imitacinio PKA modelio darbo rezultatus, būtina įsitikinti, kad kiekviena realizuoto modelio dalis veikia korektiškai. Tam tikslui, naudojantis specifikacija, buvo sukurta aibė testinių atvejų, skirtų kiekvienam modelio agregatui, kurių pagalba buvo patikrintas agregatų veikimo korektiškumas. Testų metu, agregatams, priklausomai nuo jų realizacijos, buvo pateikiami jiems būtini duomenys ir laukiama adekvačių rezultatų.

3.5.1. Skausmo generatoriaus A testavimas

Skausmo generatoriaus A veikimas yra valdomas modelio logikos. Pats agregatas, gavęs užklausimą, grąžina laiko, kuris turi praeiti vaistų dozės pareikalavimo momento, reikšmę. Kadangi reikšmėms generuoti naudojamas atsitiktinis dydis - galima tik patikrinti laiko reikšmių pasiskirstymą priklausomai nuo nurodytojo vidutinio užklausų kiekio per valandą - λ. Šiam reikalui buvo sugeneruota po 1000 laiko reikšmių prie skirtingų λ reikšmių intervale [1; 10].



Pav. 12 Vidutinio laukimo laiko priklausomybė nuo λ

Pagal gautuosius rezultatus, galima teigti, kad sugeneruotųjų laiko reikšmių vidurkiai atitinka numatytąsias tendencijas, tai yra, didėjant užklausų kiekiui, trumpėja laiko intervalai tarp užklausų.

3.5.2. Skausmo generatoriaus B testavimas

Kaip ir skausmo generatoriaus A atveju, skausmo generatorius B taip pat yra valdomas modelio logikos. Darbo pradžioje, nustatoma minimalaus vaistų koncentracijos lygis C_{min} , minimali ir maksimali galimos paciento laukimo laiko reikšmės T_{min} ir T_{max}. Užklausimo metu, agregatas gauna esamą vaistų koncentracijos lygį C. Jei C_{min} yra daugiau už C, tuomet generatorius grąžina atsitiktinę laiko reikšmę intervale [T_{min} , T_{max}].

Pirmuoju testu buvo patikrinta, ar generatorius grąžina laiko reikšmes pasiekus ir viršijus $C_{min.}$

C_{min} = 50, T_{min} = 1, T_{max} = 5

Duotoji C reikšmė	Lauktoji laiko reikšmė	Gautoji laiko reikšmė

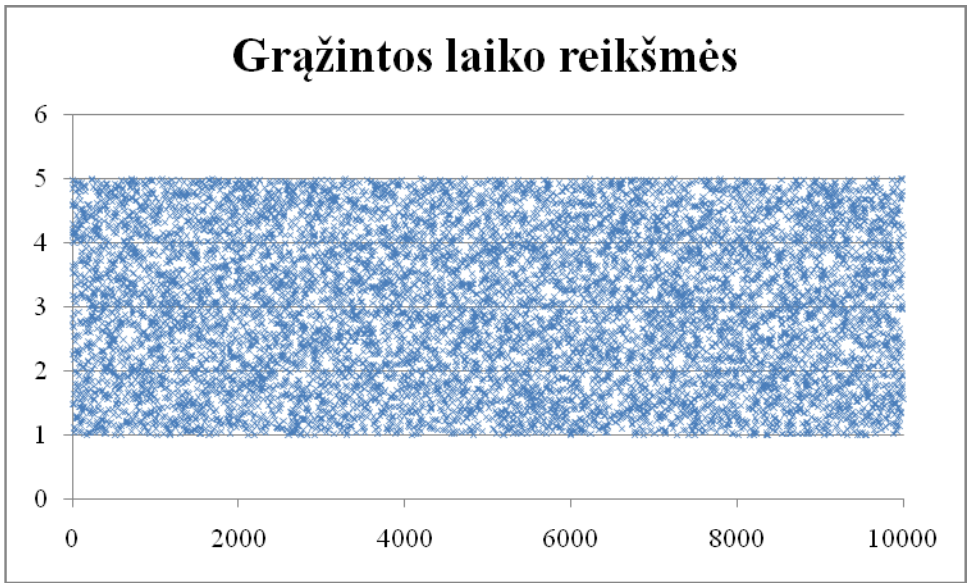
40	Bet koks skaičius intervale [T_{min} , T_{max}]	4,49095
49,1	Bet koks skaičius intervale [T_{min} , T_{max}]	1,81704
50	-1	-1
50,1	-1	-1
60	-1	-1

Lentelė 1. Pirmojo skausmo generatoriaus B testo rezultatai

Matome, kad C esant 50 ir daugiau, grąžinamas laikas lygus -1, kas signalizuoja modelio valdymo logikai, kad tą laiko reikšmę reikia ignoruoti.

Antrojo testo metu buvo sugeneruota 10000 laiko reikšmių, siekiant patikrinti, ar gautosios reikšmės nepatenka už leistino intervalo ribų.

C = 40, T_{min} = 1, T_{max} = 5



Pav. 13 Grąžintos laiko reikšmės

Pagal gautus rezultatus, matome, kad grąžintosios laiko reikšmės pasiskirsčiusios testuotame laiko intervale.

3.5.3. PKA pompos testavimas

PKA pompos agregatui, prieš pradedant darbą, būtina nustatyti pompos aktyvacijos trukmę T_{akt}. Užklausimo metu, agregatui perduodama esamo laiko reikšmė. Jei agregatas yra neaktyvioje būsenoje, jis pereina į aktyvią būseną ir grąžina naują laiko reikšmę T, kuomet vėl

taps neaktyvus. Kitu atveju, grąžinamas ankstesnės užklausos metu paskaičiuota perėjimo į neaktyvią būseną laiko momento reikšmė T.

Testo metu, buvo pateikiami laiko momentai, kuomet agregatas yra neaktyvioje būsenoje, aktyvioje būsenoje ir kuomet pereina iš aktyvios į neaktyvią.

$T_{akt} = 10$

Užklausimo momentas	Numatomas perėjimo į neaktyvią būseną laiko momentas	Agregato būsena užklausimo momentu (0 -neaktyvi, 1 - aktyvi)
0	10	0
5	10	1
9	10	1
11	21	0
15	21	1
21	31	0
25	31	1
35	45	0

Lentelė 2. PKA pompos testų rezultatai

Matome, kad agregatas būdamas aktyvioje būsenoje ir gavęs užklausą, veikia taip, kaip ir buvo tikėtasi - ignoruoja gautą užklausą.

3.5.4. IPKA pompos testavimas

IPKA pompos agregatui, kaip ir anksčiau minėtajam PKA pompos agregatui, prieš pradedant darbą, nustatoma pompos aktyvacijos trukmę T_{akt}. metu, agregatui perduodama esamo laiko reikšmė. Jei agregatas yra neaktyvioje būsenoje, jis pereina į aktyvią būseną ir grąžina naują laiko reikšmę T, kuomet vėl taps neaktyvus. Kitu atveju, vėl skaičiuojamas naujas perėjimo į neaktyvią būseną laiko momentas T.

Testo metu, buvo pateikiami laiko momentai, kuomet agregatas yra neaktyvioje būsenoje, aktyvioje būsenoje ir kuomet pereina iš aktyvios į neaktyvią.

$T_{akt} = 10$

Užklausimo momentas	Numatomas perėjimo į neaktyvią būseną laiko momentas	Agregato būsena užklausimo momentu (0 -neaktyvi, 1 - aktyvi)
0	10	0
5	15	1
9	19	1

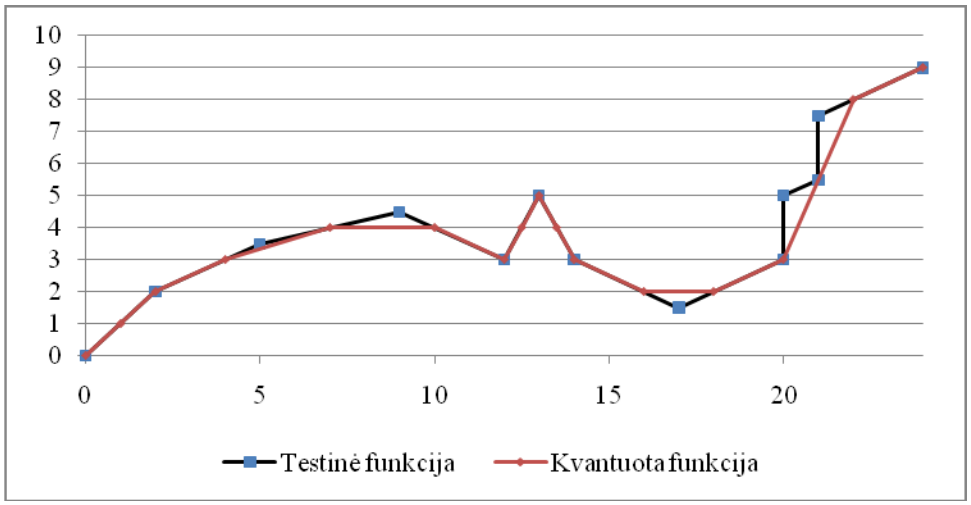
21	31	0
25	35	1
35	45	0
40	50	1
51	61	0

Lentelė 3. IPKA pompos testų rezultatai

Testo rezultatai rodo, kad agregatas veikia kaip ir buvo tikėtasi. Gavus naują užklausą aktyvios būsenos metu, pompa pratęsia savo darbą papildomai T_{akt} trukme nuo paskutiniosios užklausos gavimo.

3.5.5. Integratoriaus testavimas

Siekiant patikrinti ar integratoriaus agregatas veikia korektiškai, buvo sudarytas testas, kuomet pasirinktais laiko momentais integratoriui buvo perduodama nauja išvestinės reikšmė. Testavimo metu buvo pasirinktas diskretizacijos lygis $\Delta Q = 1$. Tuo pačiu testu, buvo patikrintas ir šuolinis funkcijos reikšmės padidinimas, kai šuolis yra $2*\Delta Q$. Gauti rezultatai (pav. 14) rodo, kad integratorius tinkamai atlieka savo funkcijas, t.y. visos gautos funkcijos reikšmės yra diskrečios ir maksimalus nukrypimas nuo testinės funkcijos reikšmių nesiekia ΔQ.



Pav. 14 Testinės ir kvantuotos funkcijų grafikų palyginimas

3.5.6. Sumatoriaus X1 testavimas

Sumatorius X1 turi tris įvestis: x1, x2 ir x3. Pradinėje būsenoje visos įvestys lygios 0. Įvesties reikšmė visą laiką išlieka pastovi, kol į ją nepaduodamas naujas signalas. Pasikeitus bent vienai įvesties reikšmei, sumatorius iškarto suskaičiuoja išėjimo reikšmę pagal formulę:

$Y = X_1^* \cdot \beta_1 + X_2^* \cdot \beta_2 + X_3^* \cdot \beta_3$ (5), kai

$\beta_1 = -k12 - k13 - kel, \beta_2 = k21, \beta_3 = k31$ (6)

Testavimui buvo sudaryti testiniai atvejai, kuomet į vieną ar kelis įėjimus buvo paduodamos iš anksto numatytos įvesties reikšmės. Testo metu buvo naudojami tokie koeficientai: *k12 = 0, k13 = 0, kel = -10, k21 = 100, k31 = 1000.*

Pateikiami testavimo metu gauti rezultatai, kurie rodo, kad sumatorius veikia korektiškai:

x1	x2	x3	y	lauktas y
1	0	0	10	10
1	1	0	110	110
1	1	1	1110	1110
0	0	1	1000	1000
0	0	0	0	0
0	1	1	1100	1100
0	0	0	0	0
1	1	1	1110	1110

Lentelė 4. Pirmo sumatoriaus testavimo rezultatai

3.5.7. Sumatoriaus X2 testavimas

Sumatorius X2 turi dvi įvestis: x1, x2. Pradinėje būsenoje visos įvestys lygios 0. Įvesties reikšmė visą laiką išlieka pastovi, kol į ją nepaduodamas naujas signalas. Pasikeitus bent vienai įvesties reikšmei, sumatorius iškarto suskaičiuoja išėjimo reikšmę pagal formulę:

$Y = X_1^* \cdot \beta_1 + X_2^* \cdot \beta_2$ (7), kai

$\beta_1 = k12, \beta_2 = -k21$ (8).

Testavimui buvo sudaryti testiniai atvejai, kuomet į vieną ar kelis įėjimus buvo paduodamos iš anksto numatytos įvesties reikšmės. Testo metu buvo naudojami tokie koeficientai: *k12 = 100, k21 = 10.*

Pateikiami testavimo metu gauti rezultatai, kurie rodo, kad sumatorius veikia korektiškai:

x1	x2	y	lauktas y
0	0	0	0

1	0	100	100
1	1	90	90
0	0	0	0

Lentelė 5. Antro sumatoriaus testavimo rezultatai

3.5.8. Sumatoriaus X3 testavimas

Sumatorius X3 turi dvi įvestis: x1, x2. Pradinėje būsenoje visos įvestys lygios 0. Įvesties reikšmė visą laiką išlieka pastovi, kol į ją nepaduodamas naujas signalas. Pasikeitus bent vienai įvesties reikšmei, sumatorius iškarto suskaičiuoja išėjimo reikšmę pagal formulę:

$Y = X_1^* \cdot \beta_1 + X_3^* \cdot \beta_3$ (9), kai

$\beta_1 = k13, \beta_3 = -k31$ (10).

Testavimui buvo sudaryti testiniai atvejai, kuomet į vieną ar kelis įėjimus buvo paduodamos iš anksto numatytos įvesties reikšmės. Testo metu buvo naudojami tokie koeficientai: *k13 = 100, k31 = 10.*

Pateikiami testavimo metu gauti rezultatai, kurie rodo, kad sumatorius veikia korektiškai:

x1	x2	y	lauktas y
0	0	0	0
1	0	100	100
1	1	90	90
0	0	0	0

Lentelė 6. Trečio sumatoriaus testavimo rezultatai

3.6. Imitacinio modeliavimo rezultatai

Siekiant patvirtinti arba paneigti darbo pradžioje suformuluotus teiginius apie PKA ir IPKA algoritmus, naudojant imitacinį PKA modelį buvo atlikti eksperimentiniai bandymai, kurių metu gauti duomenys padės atsakyti į rūpimus klausimus. Kadangi suformuluotos prielaidos liečia skirtingus aspektus, bandymų metu buvo naudojamos kelios imitacinio modelio variacijos (žiūr. skyrių 3.1.4 Agregatinis PKA modelis).

Eksperimentų metu, kuomet buvo stebimas vaistų koncentracijos pasiskirstymo paciento organizme kitimas, buvo naudojami šie imitacinio modelio parametrai :

- kel=0.158, k12=0.385, k13=0.233, k21=0.228, k31=0.021,
- pirmosios sekcijos tūris - V=13l,
- vaistų dozė - 1 mg,
- pompos aktyvios būsenos laikas – 10 min.,
- kvantinio lygio dydis – ΔQ=1. [5]

Siekiant imituoti IPKA algoritmo metu vykstantį pastovų vaistų leidimą, vaistų dozė buvo padalinta į 10 dalių ir suleidžiama atitinkamais laiko tarpais pompos aktyvios būsenos metu.

3.6.1. Skausmo generatoriaus A scenarijus

Siekiant sužinoti, kaip bendruoju atveju kinta pompos algoritmų rezultatai bei vaistų koncentracija paciento organizme, priklausomai nuo vaistų dozės pareikalavimo kiekio valandos bėgyje, eksperimentams vykdyti naudojome imitacinio modelio variantą su skausmo generatoriumi A (pav. 9). Tai suteikė galimybę abiejų pompos algoritmų atvejais naudoti tuos pačius vaisto dozės pareikalavimo laiko momentus.

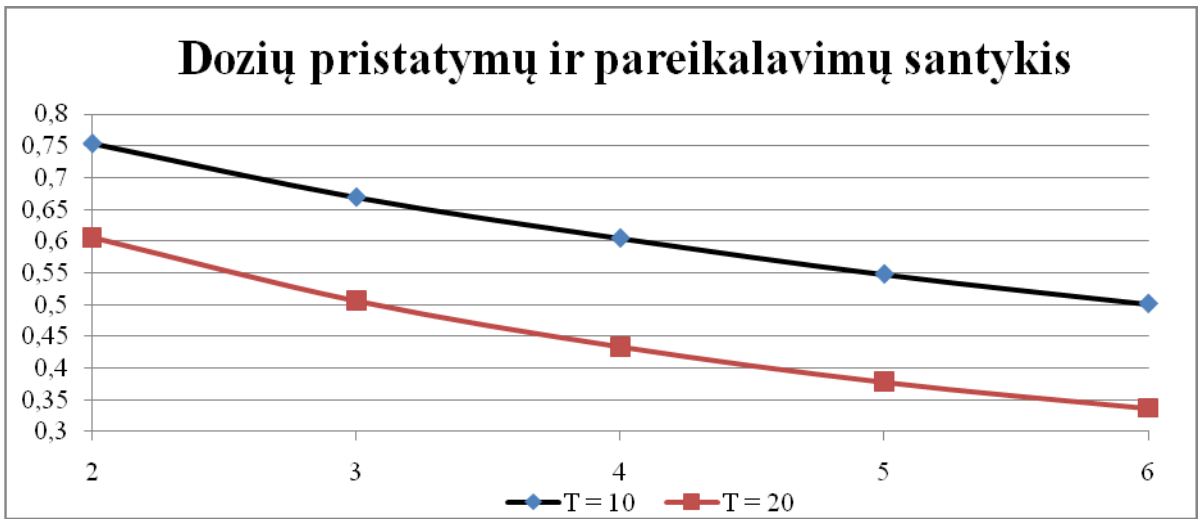
Pirmiausiai buvo atlikti statistinės analizės bandymai, siekiant nustatyti pompos algoritmų metrikų kitimą priklausomai nuo užklausų kiekio. Tam reikalui buvo atliekama po 1000 imitacijų, kurių trukmė T = 24 valandos, su skirtingais užklausų per valandą kiekiais λ, kai λ kinta sveikųjų skaičių intervale [2; 6].

Sekantys eksperimentai buvo atliekami su šiais parametrais:

- imitacijos trukmė - 24 valandos,
- užklausų kiekis per valandą $\lambda = \{2, 4, 6\}$,
- vaisto dozė {1, 2} mg,
- aktyvios būsenos trukmė T = {10, 20} min.

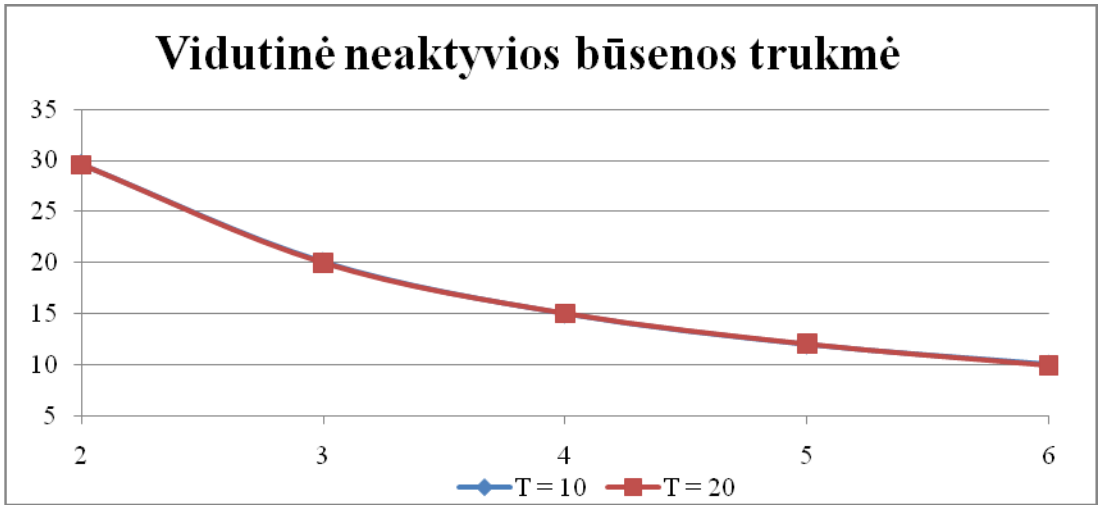
3.6.1.1. PKA pompos elgsenos tyrimai

Gautieji rezultatai rodo, kad didėjant naujos dozės pareikalavimų per valandą kiekiui, sumažėja galimybė gauti prašomą dozę. Tai tampa dar akivaizdžiau, kuomet padidinama pompos aktyvios būsenos trukmė (pav. 15).



Pav. 15 PKA pompos elgsena: dozių pristatymų ir pareikalavimų santykis

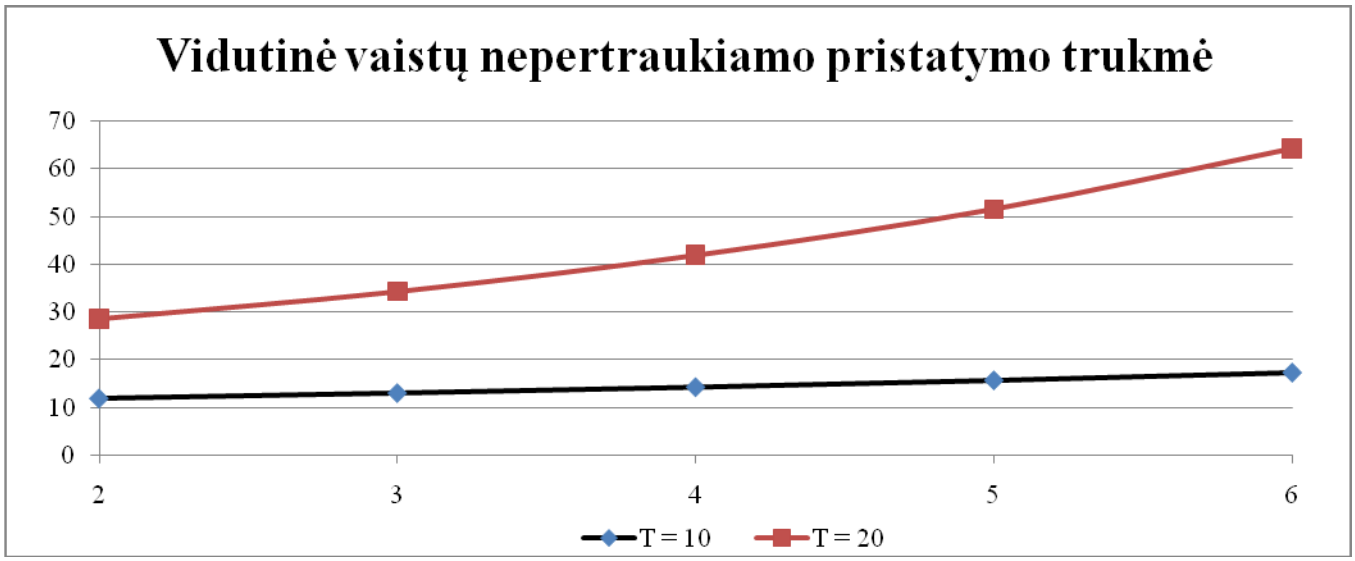
Taip pat pastebime, kad aktyvios būsenos trukmė neturi jokios įtakos vidutiniam laiko tarpui, kuomet pompa neturės darbo.



Pav. 16 PKA pompos elgsena: vidutinė neaktyvios būsenos trukmė

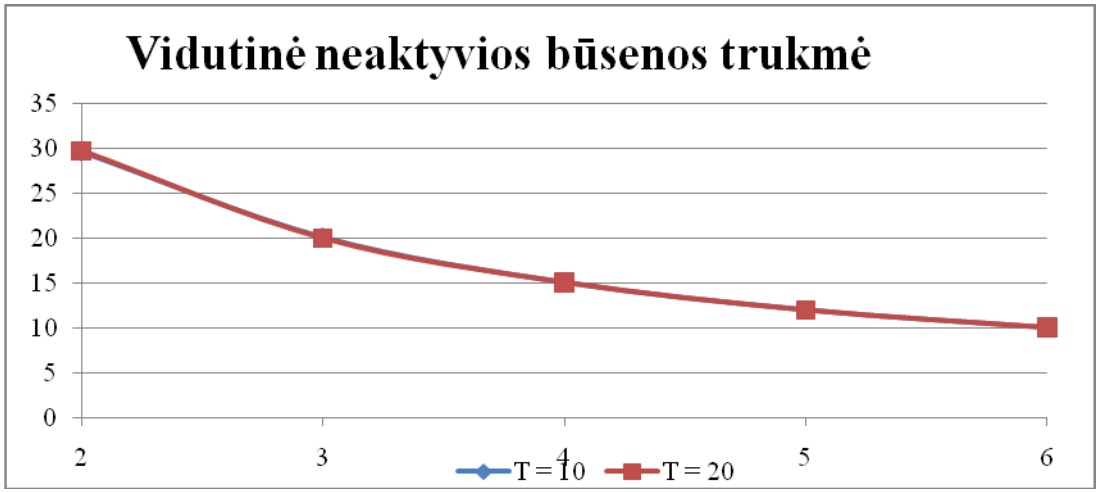
3.6.1.2. IPKA pompos elgsenos tyrimas

Dėl IPKA algoritmo skirtumų lyginant su PKA algoritmu, IPKA pompa visuomet pristatys pareikalautą vaistų dozę. Tačiau, priklausomai nuo pareikalavimų kiekio per valandą, priklauso ir vidutinė nepertraukiamo vaistų pristatymo trukmė. Pastebime, kad padidinus aktyvios būsenos trukmę, didėjant užklausų kiekiui, nepertraukiamo pristatymo trukmė didėja greičiau negu esant trumpesniam aktyvios būsenos laikui (pav. 17).



Pav. 17 IPKA pompos elgsena: vidutinė vaistų nepertraukiamo pristatymo trukmė

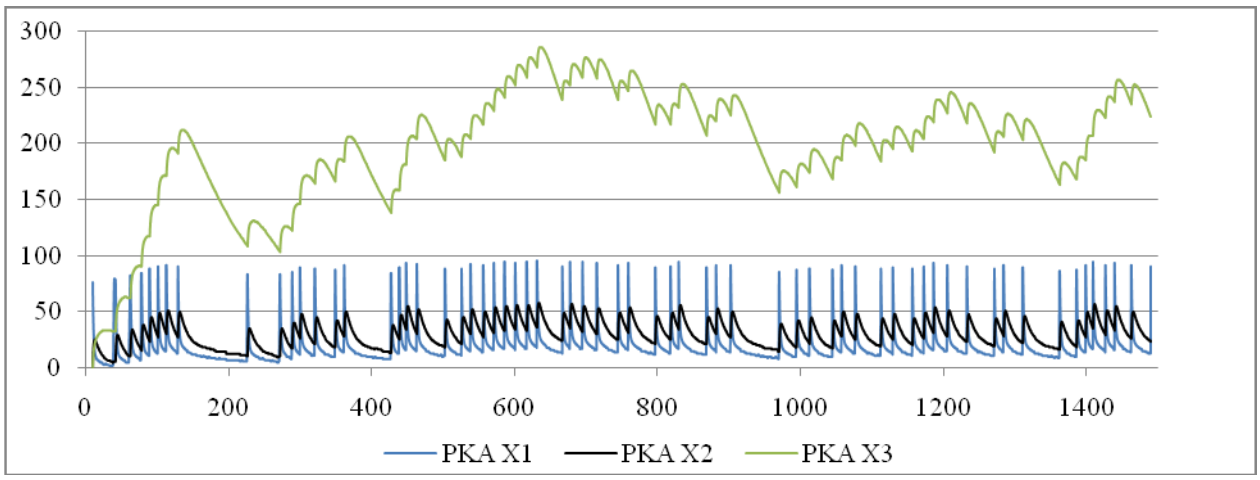
Analogiškai, kaip ir PKA algoritmo atveju, IPKA pompos aktyvios būsenos trukmė neturi įtakos buvimo neaktyvioje būsenoje laiko intervalams.



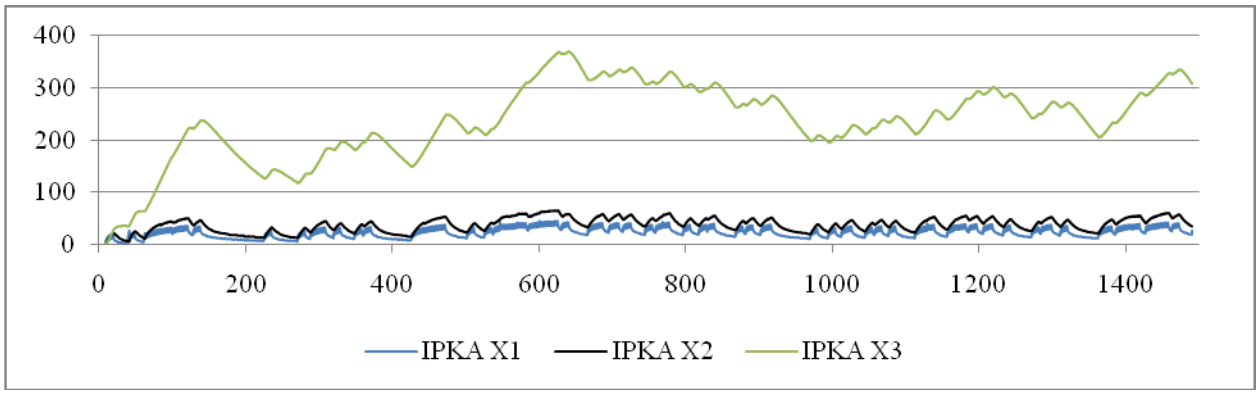
Pav. 18 IPKA pompos elgsena: vidutinė neaktyvios būsenos trukmė

3.6.1.3. Farmakokinetinio modelio tyrimas

Atlikus imitacinius eksperimentus ir apdorojus rezultatus, gavome grafikus, vaizduojančius vaistų koncentracijos kitimą atskiruose kompartamentuose. Pagal gautuosius rezultatus matome, kad PKA algoritmo atveju, vaistų koncentracija visuose kompartementuose kinta šuoliškai ties kiekvienu vaisto suleidimo momentu (pav. 19). Tuo pačiu, šuolio amplitudė, lyginant su IPKA algoritmu (pav. 20), yra didesnė, kas lemia staigesnius vaistų koncentracijos pokyčius paciento organizme.

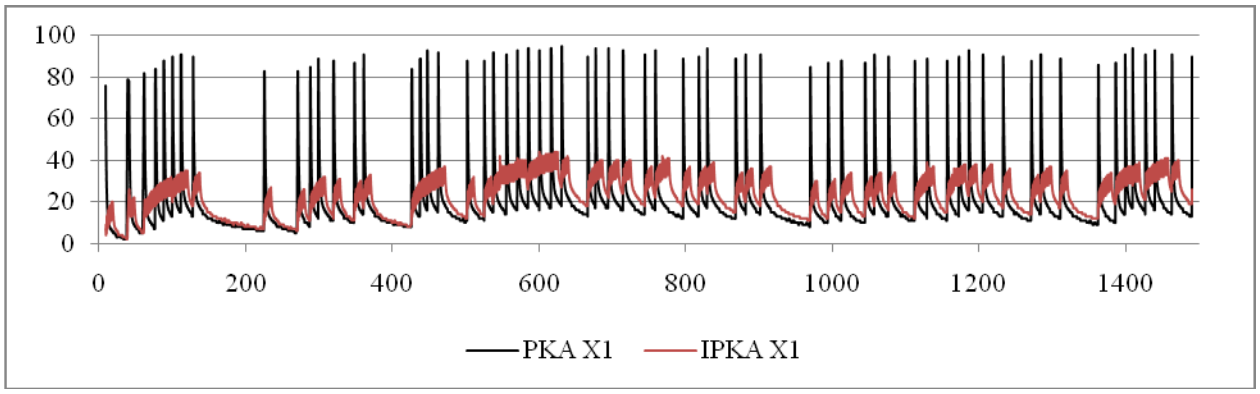


Pav. 19 Eksperimento rezultatai (parametrai: imitacijos trukmė - 24 val., T= 10 min., λ = 4, dozė = 1 mg, PKA pompa)

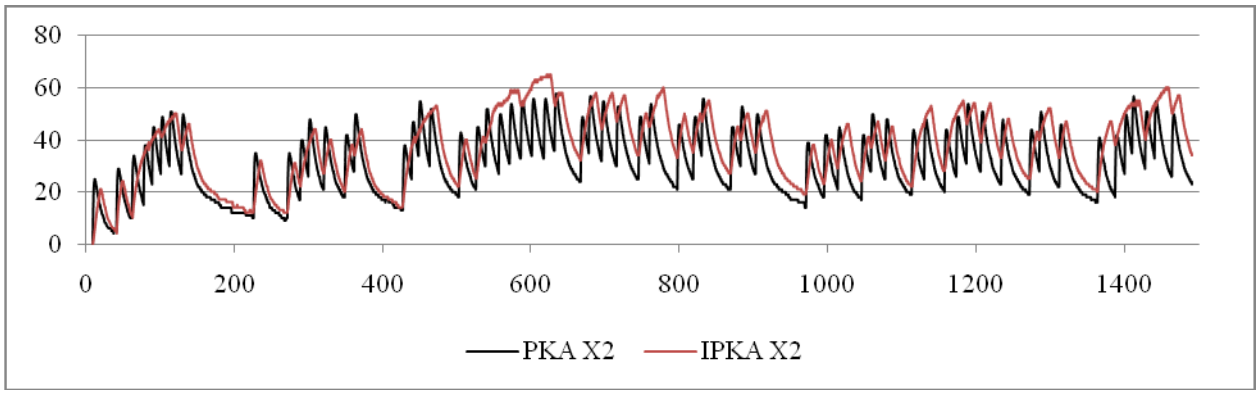


Pav. 20 Eksperimento rezultatai (parametrai: imitacijos trukmė - 24 val., T= 10 min., λ = 4, dozė = 1 mg, IPKA pompa)

Kadangi IPKA algoritmo atveju vaistai leidžiami tolygiai, jų koncentracija tiek pirmajame (pav. 21), tiek ir antrajame kompartamente (pav. 22) kinta tolygiau ir išlieka pastovesnė, lyginant su PKA algoritmu. Tiesa, kadangi IPKA pompa aktyvios būsenos metu neatmeta naujo užklausimo, o pratęsia vaistų suleidimą, šių eksperimentų metu buvo gausu atvejų, kai antrajame kompartamente IPKA algoritmu suleistų vaistų koncentracija tapdavo didesnė už PKA vaistų koncentraciją. Dėl šios priežasties, gauti rezultatai nėra tinkami vertinti algoritmų pagal pasiektus vaistų koncentracijos lygius (eksperimentų metu nesistengta imituoti realios paciento elgsenos).

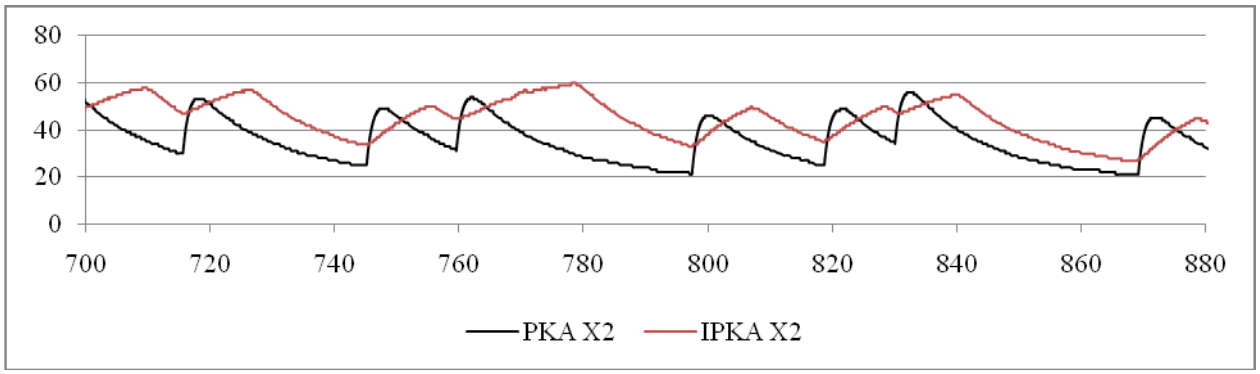


Pav. 21 Koncentracijos kitimo pirmajame kompartamente palyginimas (parametrai: imitacijos trukmė - 24 val., T= 10 min., λ = 4, dozė = 1 mg)

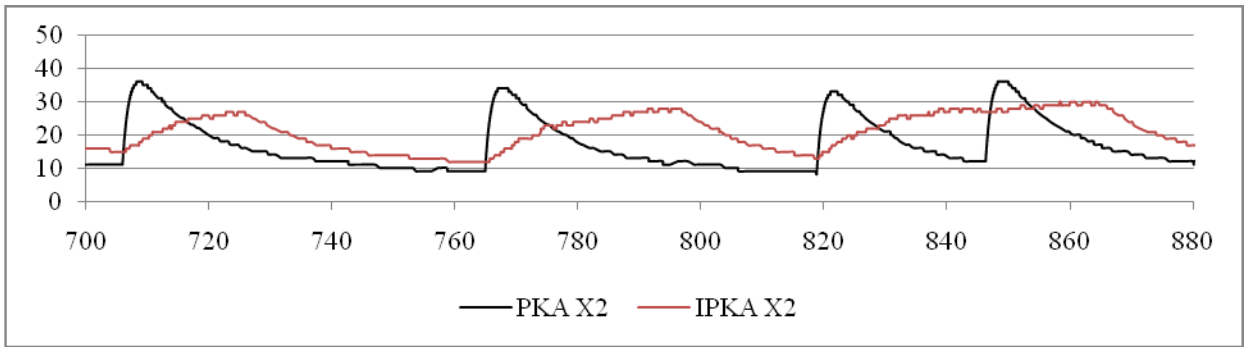


Pav. 22 Koncentracijos kitimo antrajame kompartamente palyginimas (parametrai: imitacijos trukmė - 24 val., T= 10 min., λ = 4, dozė = 1 mg)

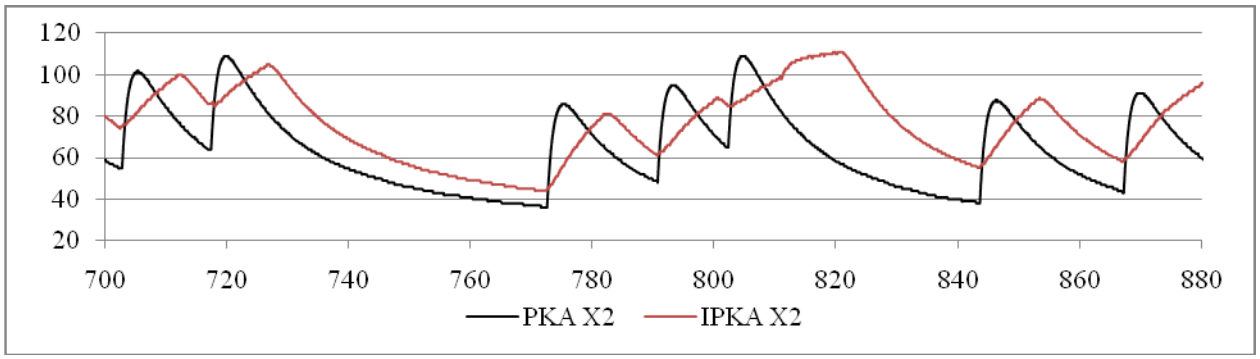
Taip pat buvo pastebėta, kad IPKA algoritmas leidžia keisti vaistų koncentracijos kitimo spartą, keičiant vaitų dozės dydį arba pompos aktyvios būsenos trukmę (pav. 23, 24, 25). Pastebime, kad PKA algoritmu suleistų vaistų koncentracijos šuolio pokyčiui įtaką galime daryti tik keisdami suleidžiamų vaistų kiekį.



Pav. 23 Koncentracijos kitimo antrajame kompartamente palyginimo fragmentas (parametrai: imitacijos trukmė - 24 val., T= 10 min., λ = 4, dozė = 1 mg)



Pav. 24 Koncentracijos kitimo antrajame kompartamente palyginimo fragmentas (parametrai: imitacijos trukmė - 24 val., T= 20 min., λ = 4, dozė = 1 mg)



Pav. 25 Koncentracijos kitimo antrajame kompartamente palyginimo fragmentas (parametrai: imitacijos trukmė - 24 val., T= 10 min., λ = 4, dozė = 2 mg)

3.6.2. Skausmo generatoriaus B scenarijus

Naudojant imitacinio modelio variacijas su skausmo generatoriumi B (pav. 10), buvo siekiama nustatyti, kaip kinta vaistų koncentracija antrajame paciento organizmo kompartamente. Eksperimentai buvo atliekami su tokiais parametrais:

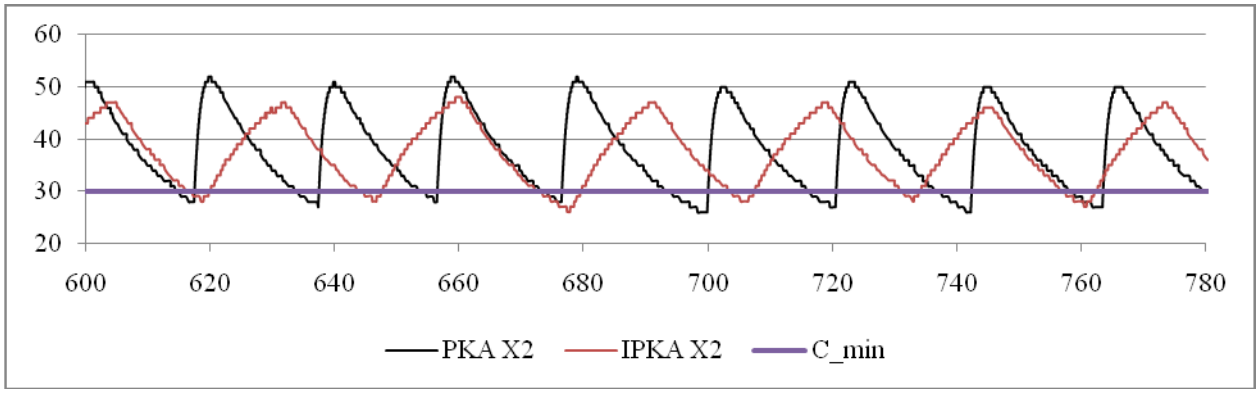
- imitacijos trukmė - 24 valandos,
- norimas palaikyti koncentracijos lygis C_{min} = {30, 55, 80) ng/ml,
- vaistų dozė {1, 2} mg,
- pompos aktyvios būsenos laikas T = {10, 20} min.,
- laikas iki naujos dozės pareikalavimo $T_{reik.}$ = [1;5] min.

Apdorojus surinktus duomenis, gavome vaistų koncentracijos kitimo paciento organizme grafikus. Apibendrinant gautus rezultatus, pastebime, kad IPKA algoritmas esant per didelei vaitų dozei prie tam tikros siekiamos palaikyti vaistų koncentracijos arba per trumpam aktyvios būsenos laikui, negali pasigirti ženkliu pranašumu prieš įprastą PKA algoritmą. Tai akivaizdžiai matosi pav. 26: siekiant palaikyti žemą koncentracijos lygį, dėl per didelės dozės, IPKA algoritmo koncentracijos perviršiai prilygsta PKA algoritmui. Būtina atkreipti dėmesį, kad nenagrinėjame vaistų suvartojimo kiekio ir vidutinių perviršio reikšmių. Siekiant sumažinti IPKA koncentracijos perviršius nekeičiant vaisto dozės, galimi du variantai:

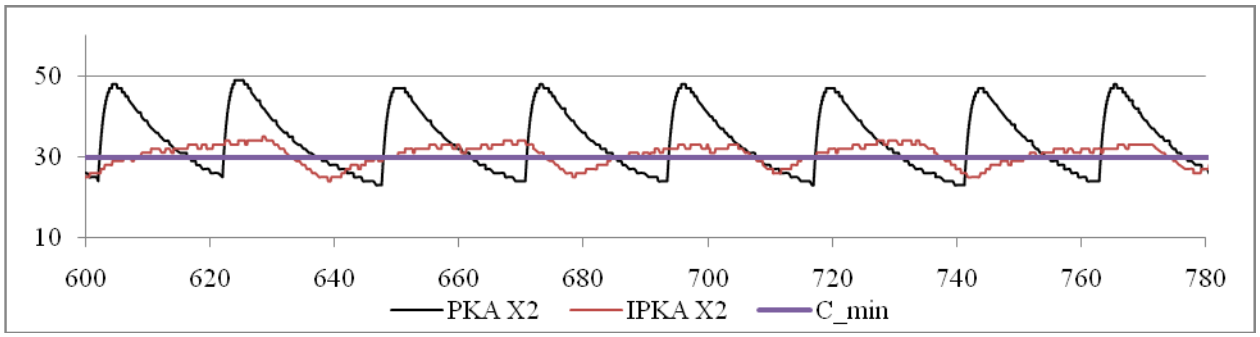
padidinti pompos aktyvios būsenos trukmę (pav. 27);

padidinti norimą palaikyti koncentracijos lygį (pav. 28).

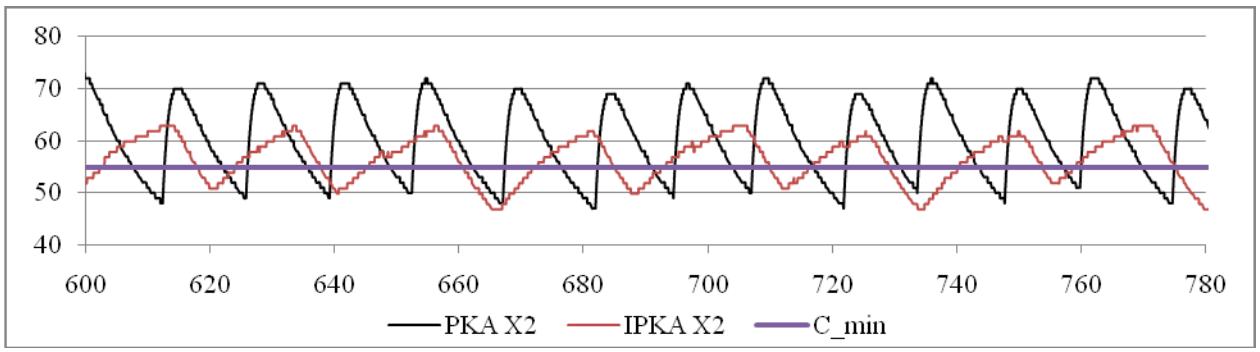
Abiem atvejais pastebime, kad pagerėja tik IPKA algoritmo rezultatai.



Pav. 26 Koncentracijos kitimo antrajame kompartamente palyginimo fragmentas (parametrai: imitacijos trukmė - 24 val., T= 10 min., C = 30 ng/ml, dozė = 1 mg)

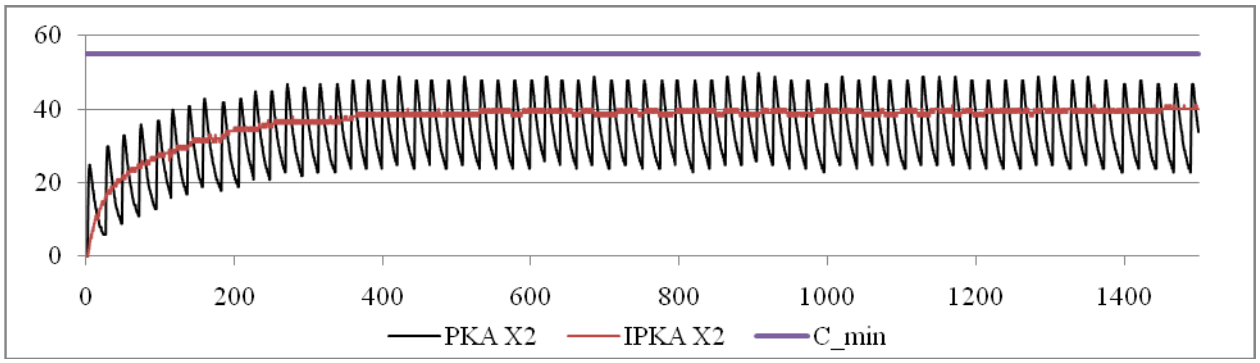


Pav. 27 Koncentracijos kitimo antrajame kompartamente palyginimo fragmentas (parametrai: imitacijos trukmė - 24 val., T=20 min., C = 30 ng/ml, dozė = 1 mg)



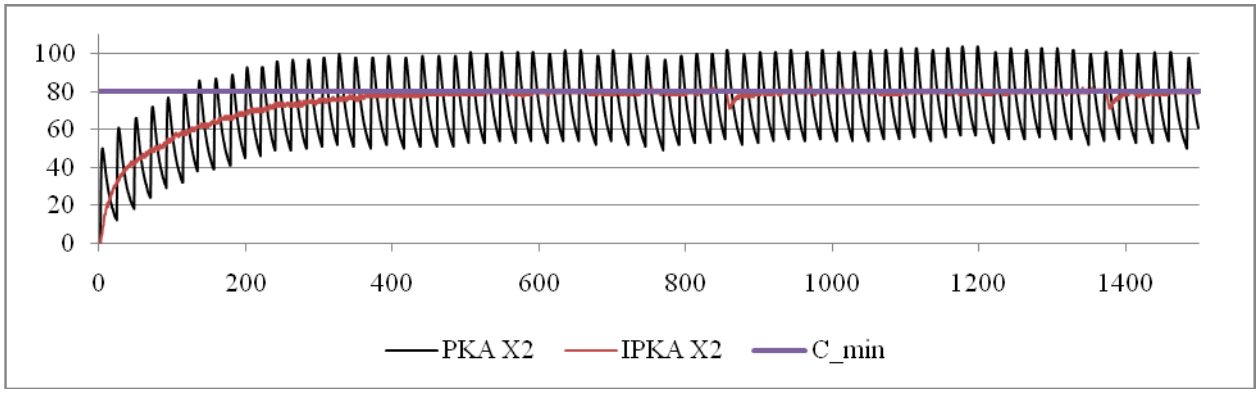
Pav. 28 Koncentracijos kitimo antrajame kompartamente palyginimo fragmentas (parametrai: imitacijos trukmė - 24 val., T= 10 min., C = 55 ng/ml, dozė = 1 mg)

Taip pat esama atvejų, kai dėl netinkamai parinktos vaisto dozės, aktyvios būsenos trukmės, norimo palaikyti koncentracijos lygio parametrų, neįmanoma pasiekti norimo koncentracijos lygio (pav. 29), nes suleistas vaistas dėl nepakankamo kiekio iš organizmo pašalinamas greičiau negu pasiekiama norima koncentracija. Tačiau net ir tokiu atveju, naudojant IPKA algoritmą, paciento juntamas skausmas kinta gerokai tolygiau negu PKA atveju.

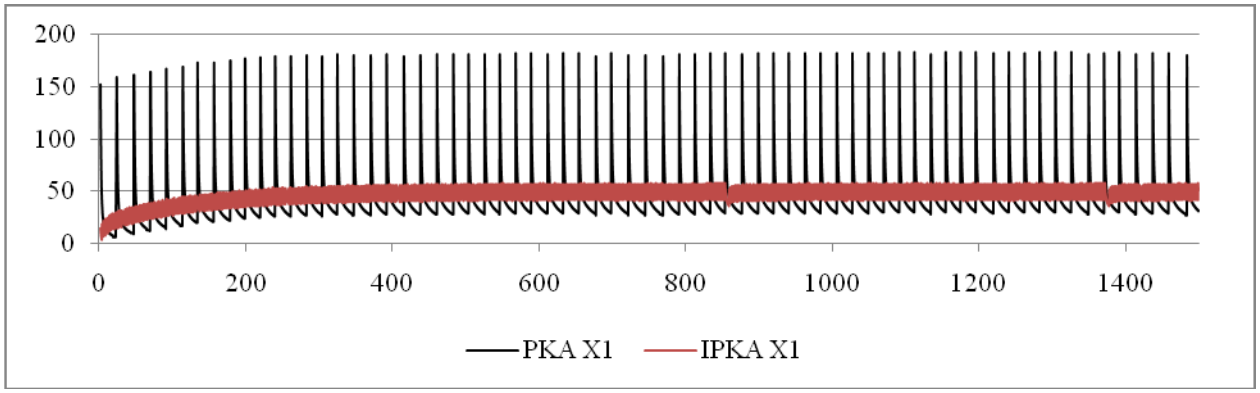


Pav. 29 Koncentracijos kitimo antrajame kompartamente palyginimas (parametrai: imitacijos trukmė - 24 val., T= 20 min., C = 55 ng/ml, dozė = 1 mg)

Tinkamai suderinus parametrus, IPKA algoritmo pagalba galima gauti rezultatus, kurie rodo, kad paciento skausmo lygis yra sumažinamas iki beveik nejuntamo, o vaistų koncentracijos lygis palaikomas beveik be nukrypimų, kas tiesiog neįmanoma pasiekti naudojant PKA algoritmą (pav. 30). Tiesa, būtina atkreipti dėmesį į tai, kad tokiu atveju IPKA pompa vaistus leidžia beveik be pertrūkių (pav. 31). Dėl šios priežasties atsiranda rizika, kad pacientas su laiku taps nepakantus net ir mažiausiam skausmui.



Pav. 30 Koncentracijos kitimo antrajame kompartamente palyginimas (parametrai: imitacijos trukmė - 24 val., T= 20 min., C = 80 ng/ml, dozė = 2 mg)



Pav. 31 Koncentracijos kitimo pirmajame kompartamente palyginimas (parametrai: imitacijos trukmė - 24 val., T= 20 min., C = 80 ng/ml, dozė = 2 mg)

4. IŠVADOS

Sudarius imitacinį PKA modelį ir atlikus aibę eksperimentinių bandymų, pagal surinktus duomenis, galime daryti išvadą, kad darbo pradžioje suformuluotos hipotezės, iš esmės, pasitvirtino.

Kaip ir buvo tikėtasi, įprastinis PKA algoritmas neleidžia pacientui viršyti numatytos vaistų dozės, kadangi papildomų vaistų dozių pristatymo tikimybė mažėja sulig prašymo kiekio didėjimu. IPKA pompos atveju vaisto dozių prašymo kiekio didėjimas lemia ilgesnį laiko tarpą, kuomet leidžiami vaistai. Tuo pačiu padidėja ir gaunamų vaistų kiekis, nes vaistai leidžiami pastoviu tempu.

Realioje situacijoje, kuomet pacientas prašo vaistų tik tada, kai junta skausmą (vaistų koncentracija organizme mažesnę negu norima palaikyti), išryškėja pagrindinis IPKA algoritmo pranašumas - tinkamai parinkus vaistų dozės bei pompos aktyvios būsenos parametrus, dėl tolygesnio vaistų koncentracijos kitimo tampa gerokai lengviau išlaikyti vaisų koncentraciją ties norimu terapiniu lygiu. Kai kuriais atvejais įmanoma netgi sumažinti koncentracijos svyravimus iki minimumo, ko nepavyksta atlikti naudojant įprastą PKA algoritmą.

Būtina atkreipti dėmesį, kad šiame darbe nebuvo atliekamas algoritmų palyginimas pagal vaistų kiekio sunaudojimą bei statistinė vaistų koncentracijos kitimo analizė. Tai galima būtų atlikti patobulinus sukurtą modeliavimo aplinką. Taip pat, atlikus imitacinio modelio patobulinimus, galima būtų atlikti tyrimus imituojant kelių skirtingo poveikio vaistų naudojimą.

5. NAUDOTA LITERATŪRA

[1] A.M. Chang PhD RN, W.Y. Ip BN MPhil RN RM and T.H. Cheung MBBS FRCOG, *Patient-controlled analgesia versus conventional intramuscular injection: a cost effectiveness analysis*. JAN, 2003, 531-541.

[2] E. Kofman. *Discrete Event Based Simulation and Control of Continuous Systems: daktaro disertacija*, Universidad Nacional de Rosario, Argentina, 2003.

[3] E. Kofman, M. Lapadula, and E. Pagliero, *PowerDEVS: A DEVS–Based Environment for Hybrid System Modeling and Simulation*. Technical Report LSD0306, LSD, Universidad Nacional de Rosario, Argentina, 2003.

[4] Henrikas Pranevičius, *Sudėtingų sistemų formalizavimas ir analizė*. Kauno technologijos universitetas, 2008. Psl. 13-18.

[5] H. Pranevicius, M. Pranevicius, O. Pranevicius, D. Makackas, K. Sutiene**.** *Pharmacokinetics of Infusion Pca (ipca).* World institute of pain 5th world congress, New York, USA, 2009, March 13-16.

[6] H.F. Hill, A.M. Mackie, B.A. Coda, K. Iverson, C.R. Chapman. *Patient-controlled analgesic administration. A comparison of steady-state morphine infusions with bolus doses*. Cancer, John Wiley & Sons, Inc., 1991, 67, 873–882.

[7] Janie F. Franz Jennifer Lee Losey, RN. *Patient-controlled analgesia*. [interaktyvus]. Encyclopedia of Surgery. [žiūrėta 2011-04-30]. Prieiga per internetą: < http://www.surgeryencyclopedia.com/La-Pa/Patient-Controlled-Analgesia.html >

[8] M. Pranevicius, O. Pranevicius, H. Pranevicius, L. Simaitis. *Piece-Linear Aggregates for Formal Specification and Simulation of Hybrid Systems: Pharmacokinetics Patient-Controlled Analgesia.* Electronics and Electrical Engineering. – Kaunas: Technologija, 2011. – No. 4(110). – P. 129-132.

[9] Samuel Uretsky, PharmD. *Analgesics. Side effects.* [interaktyvus]. Encyclopedia of Surgery. [žiūrėta 2011-04-30]. Prieiga per internetą: < http://www.surgeryencyclopedia.com/A-Ce/Analgesics.html >

6. TERMINŲ IR SANTUMPŲ ŽODYNAS

Analgezija - skausmo nejautimas, atsirandantis dėl nervų sistemos ligų arba sukeliamas dirbtinai.

Atkarpomis tiesinis agregatas - tai sistemų formalizavimui naudojamas automatų modelių klasei priklausantis objektas, aprašomas nurodant būsenų aibę *Z*, įėjimo signalų aibę *X*, išėjimo signalų aibę *Y*, perėjimo operatorių *H* ir išėjimo operatorių *G*. Agregato būsenos kinta apibrėžtų išorinių ir vidinių įvykių pasireiškimo metu. [1]

Bolusas - toks gydymo medikamentais būdas, kurio metu vaisto koncentraciją kraujyje siekiama padidinti iki tam tikros efektyvumo ribos.

Farmakokinetika - farmakologijos šaka, nagrinėjanti procesus, susijusius su farmakologinėmis medžiagomis organizme, tokius kaip jų absorbciją, pasiskirstymą, metabolizmą ir likvidavimą.

Infuzija - tai tirpalo leidimas srovele į adata ar kateteriu punktuotą veną.

Kompartamentas - sudėtinė objekto dalis, į kurias jis yra padalintas. Žmogaus organizmo atveju, kompartamentu galima laikyti vieną kurią nors kūno sritį ar organų sistemą.

Kvantuota reikšmė - pagal diskretizavimo tinklelį kvantuotos funkcijos reikšmės riba tarp dviejų gretimų jos kvantų.

Opioidas - cheminis junginys (medžiaga), veikianti opioidinius receptorius, kurie randami daugiausia centrinėje nervų sistemoje, ir turinti nuskausminamųjų savybių.

Paciento kontroliuojama analgezija (PKA) - (angl. *Patient-Controlled Analgesia - PCA*) - bet koks skausmo malšinimo metodas, kuriuo savo juntamo skausmo lygį valdo pats pacientas savarankiškai, pagal jo suvokiamą savijautą ir poreikius.

QSS modelis - (*Quantized State System, liet. kvantuotos būsenos sistema)* modelis, aprašantis netiesinių funkcijų diskretizavimą.

7. PRIEDAI

Imitacinio PKA modelio realizacijos išeities tekstai:

7.1. Failas integrator.lua

```
--[[*********************************************************************
        ****ZEMIAU ESANCIO TEKSTO NEKEISTI*************************************]]--
function exists(n)
                local f = io.open(n)
                if f == nil then
                        return false
                else
                        f = io.close()
                        return true
                end
end

f_num = 0
t = true
f_name = os.time() .. "_lua_integ_log_" .. f_num .. ".txt"

while t do
        if exists(f_name) then
                f_num = f_num + 1
                f_name = os.time() .. "_lua_integ_log_" .. f_num .. ".txt"
        else
                t = false
        end
end

local file = io.open(f_name, "w")
file:write("lua pradejo darba\n")

--[[*********************************************************************
        ****ZEMIAU ESANTI TEKSTA GALIMA KEISTI**********************************]]--
local p_self

local in_names
local in_values = {}
local out_names
local out_values = {}

local delta_q   --qvantinio dydzio vienetas
local t_req_last
local flux_last
local Q_level
local sigma_last
local y_req_last
local reset_count

--*****************************************
--***HEAD PRADZIA*****************************
function head(pointer)
        file:write("integrator::head()\n")
        shape_type = {"circle"}

        in_types = {"double", "double"}
        in_names = {"epsilon", "delta_Q"}
        in_defaults = {0.1, 1.0}
        in_controls = {"text_box", "text_box"}

        out_types = {"double", "double", "double", "int", "int", "double", "double", "double"}
        out_names = {"t_req", "Q_level_rq", "flux_req", "req_type", "err", "sigma", "Q_level",
"y_req"}
        out_rep_types = {"chart", "chart", "chart", "chart", "chart", "chart", "chart", "chart"}

        c_set_shape_type(pointer, shape_type[1])
```

```
        c_set_in_types(pointer, in_types[1], in_types[2])
        c_set_in_names(pointer, in_names[1], in_names[2])
        c_set_in_defaults(pointer, in_defaults[1], in_defaults[2])
        c_set_in_controls(pointer, in_controls[1], in_controls[2])
        c_set_out_types(pointer, out_types[1], out_types[2], out_types[3], out_types[4],
out_types[5], out_types[6], out_types[7], out_types[8])
        c_set_out_names(pointer, out_names[1], out_names[2], out_names[3], out_names[4],
out_names[5], out_names[6], out_names[7], out_names[8])
        c_set_out_rep_types(pointer, out_rep_types[1], out_rep_types[2], out_rep_types[3],
out_rep_types[4], out_rep_types[5], out_rep_types[6], out_rep_types[7], out_rep_types[8])
        file:write("integrator::head() end\n")
end
--***HEAD PABAIGA*****************************
--*******************************************
function prepare(pointer)
        file:write("integrator::prepare()\n")

        p_self = pointer

        in_values[1], in_values[2], ok = c_get_init_vals(p_self, 2)
        eps = in_values[1]
        delta_q = in_values[2] -- kvanto dydis

        t_req_last = 0                          -- paskutinio uzklausimo laikas
        flux_last = 0                 -- paskutine naudota isvestine
        Q_level = 0                   -- esamas kvantinis lygis
        sigma_last = 0                -- paskutinis numatytas laiko tarpas (delta_t iki ivykio
pabaigos)
        y_req_last = 0                -- paskutinio uzklausimo metu pasiekta funkcijos reiksme
kvante (kitaip tariant, pokytis nuo apatines kvantines reiksmes arba funkcijos reiksme -
kvantinis lygis * dvanto dydis)

        reset_count = 0
        file:write("integrator::prepare() end\n")
end
--req_type 0 == isorinis ivykis
--req_type 1 == pompos ivykis
--req_type 2 == vidnis ivykis
--req_type 3 == resetinam reiksmes
--      t_req uzklausimo laiko momentas
-- flux_X uzklausimo momentu esama  funkcijos isvestine
--grazinam sigma, kvantini lygio reiksme (y, o ne kvantinio lygio numeri), isvestine, klaidos
koda
-- err ==  -1  flux == 0
--                     1      vidinis  ivykis, flux > 0
--                     2      isorinis ivykis, flux > 0, t_req == t_req_last + sigma_last
--                     3      isorinis ivykis, flux > 0, t_req  > t_req_last
--                     4      vidinis  ivykis, flux < 0
--                     5      isorinis ivykis, flux < 0, t_req == t_req_last + sigma_last
--                     6      isorinis ivykis, flux < 0, t_req  > t_req_last
--                     7      pompos ivykis (flux == 0)
--                     8   pompos ivykis (flux > 0)
--                     9      pompos ivykis (flux < 0)

function run_logic(t_req, req_type, flux)
--"t_req", "Q_level_rq", "flux_req", "req_type", "err", "sigma", "Q_level", "y_req"}
-- return new_time, flux, Q_level, err_code
        t_req = round(t_req, 4)
        flux = round(flux, 4)
        file:write("integrator::run_logic(" .. t_req .. ", " .. req_type .. ", " .. flux ..
")\n")
        file:write("integrator::t_req_last " .. t_req_last .. ", flux_last " .. flux_last .. ",
Q_level " .. Q_level .. ", sigma_last " .. sigma_last .. ", y_req_last " .. y_req_last .. "\n")
        c_update_res(p_self, out_types[1], t_req, out_names[1])
        c_update_res(p_self, out_types[2], Q_level, out_names[2])
        c_update_res(p_self, out_types[3], flux, out_names[3])
        c_update_res(p_self, out_types[4], req_type, out_names[4])

        if req_type == 0 then
```

```
                if flux == 0 then
                        flux_last = flux
                        sigma = -1
                        Q_sigma = Q_level * delta_q

                        file:write("   return: sigma: " .. sigma .. ", flux: " .. flux_last .. ",
Q: " .. Q_level * delta_q .. ", err -1 \n")
                        c_update_res(p_self, out_types[5], -1, out_names[5])
                        c_update_res(p_self, out_types[6], sigma, out_names[6])
                        c_update_res(p_self, out_types[7], Q_sigma, out_names[7])
                        c_update_res(p_self, out_types[8], 0, out_names[8])
                        file:write("integrator::run_logic() end \n\n")
                        return "double", -1, "double",  0, "double", Q_sigma, "int", -1
-------------------------------------------------------------------------------------------
                elseif flux > 0 then
                        delta_t = t_req - t_req_last
                        file:write("   delta_t " .. delta_t .. ", t_req " .. t_req .. ",
t_req_last " .. t_req_last .. "\n")
                        y_req = delta_t * flux_last
                                if flux_last < 0 and y_req_last == delta_q and y_req == 0 then
                                        y_req_last = 0
                                end
                                if flux_last > 0 and y_req_last == delta_q and y_req == 0 then
                                        y_req_last = 0
                                end
                                if flux_last == 0 and y_req_last == delta_q and y_req == 0 then
                                        y_req_last = 0
                                end
                        sigma = (delta_q - (y_req_last + y_req)) / flux
                        y_req_last = y_req_last + y_req
                        sigma = round(sigma, 4)
                        sigma_last = sigma
                        t_req_last = t_req
                        flux_last = flux
                        Q_sigma = Q_level * delta_q
                        c_update_res(p_self, out_types[5], 3, out_names[5])
                        c_update_res(p_self, out_types[6], sigma, out_names[6])
                        c_update_res(p_self, out_types[7], Q_sigma, out_names[7])
                        c_update_res(p_self, out_types[8], y_req, out_names[8])

                        file:write("   return: sigma: " .. sigma .. ", flux: " .. flux_last .. ",
Q: " .. Q_sigma .. ", err 3 \n")
                        file:write("integrator::run_logic() end \n\n")
                        return "double", sigma, "double",  flux, "double", Q_sigma, "int", 3
-------------------------------------------------------------------------------------------
                elseif flux < 0 then
                        delta_t = t_req - t_req_last
                        file:write("   delta_t " .. delta_t .. ", t_req " .. t_req .. ",
t_req_last " .. t_req_last .. "\n")
                        y_req = delta_t * flux_last
                        if flux_last > 0 and y_req_last == 0 and y_req == 0 then
                                y_req_last = delta_q
                        end
                        if flux_last < 0 and y_req_last == 0 and y_req == 0 then
                                y_req_last = delta_q
                        end
                        if flux_last == 0 and y_req_last == 0 and y_req == 0 then
                                y_req_last = delta_q
                        end
                        sigma = (y_req_last + y_req) / flux * -1
                        y_req_last = y_req_last + y_req
                        sigma = round(sigma, 4)
                        sigma_last = sigma
                        t_req_last = t_req
                        flux_last = flux
                        Q_sigma = Q_level * delta_q

                        c_update_res(p_self, out_types[5], 6, out_names[5])
                        c_update_res(p_self, out_types[6], sigma, out_names[6])
                        c_update_res(p_self, out_types[7], Q_sigma, out_names[7])
                        c_update_res(p_self, out_types[8], y_req, out_names[8])
```

```
                        file:write("   return: sigma: " .. sigma .. ", flux: " .. flux_last .. ",
Q: " .. Q_sigma .. ", err 6 \n")
                        file:write("integrator::run_logic() end \n\n")
                        return "double", sigma, "double",  flux, "double", Q_sigma, "int", 6
-------------------------------------------------------------------------------------------------
                end
        elseif req_type == 1 then
                plus_y = flux
-------------------------------------------------------------------------------------------------
                if flux_last == 0 then
                        plus_q_tmp = y_req_last + plus_y
                        y_req_last = (y_req_last + plus_y) % delta_q
                        plus_q = (plus_q_tmp - y_req_last) / delta_q
                        Q_level = Q_level + plus_q
                        sigma = 0
                        Q_sigma = Q_level * delta_q
                        y_req = 0
                        c_update_res(p_self, out_types[5], 7, out_names[5])
                        c_update_res(p_self, out_types[6], sigma, out_names[6])
                        c_update_res(p_self, out_types[7], Q_sigma, out_names[7])
                        c_update_res(p_self, out_types[8], y_req, out_names[8])
                        --file:write(" integrator::y_req " .. y_req .. ", y_req_last " ..
y_req_last .. ", y_to_go " .. delta_q - y_req_last .. ", sigma_last " .. sigma_last .."\n")
                        file:write("   return: sigma: " .. sigma .. ", flux: " .. flux_last .. ",
Q: " .. Q_sigma .. ", err 7 \n")
                        file:write("integrator::run_logic() end \n\n")
                        return "double", 0, "double",  0, "double", Q_sigma, "int", 7
-------------------------------------------------------------------------------------------------
                elseif flux_last > 0 then
                        delta_t = t_req - t_req_last
                        y_req = delta_t * flux_last
                        y_req_last = y_req_last + y_req
                        plus_q_tmp = y_req_last + plus_y
                        y_req_last = (y_req_last + plus_y) % delta_q
                        plus_q = (plus_q_tmp - y_req_last) / delta_q
                        Q_level = Q_level + plus_q
                        y_req = delta_q - y_req_last
                        sigma = y_req /flux_last
                        sigma = round(sigma, 4)
                        sigma_last = sigma
                        t_req_last = t_req
                        Q_sigma = Q_level * delta_q

                        c_update_res(p_self, out_types[5], 8, out_names[5])
                        c_update_res(p_self, out_types[6], sigma, out_names[6])
                        c_update_res(p_self, out_types[7], Q_sigma, out_names[7])
                        c_update_res(p_self, out_types[8], y_req, out_names[8])

                        file:write("   return: sigma: " .. sigma .. ", flux: " .. flux_last .. ",
Q: " .. Q_sigma .. ", err 8 \n")
                        file:write("integrator::run_logic() end \n\n")
                        return "double", sigma, "double",  flux_last, "double", Q_sigma, "int", 8
-------------------------------------------------------------------------------------------------
        elseif flux_last < 0 then
                        delta_t = t_req - t_req_last
                        y_req = delta_t * flux_last * -1
                        y_req_last = y_req_last - y_req
                        plus_q_tmp = y_req_last + plus_y
                        y_req_last = (y_req_last + plus_y) % delta_q
                        plus_q = (plus_q_tmp - y_req_last) / delta_q
                        Q_level = Q_level + plus_q
                        y_req = y_req_last
                        sigma = y_req /flux_last * -1
                        sigma = round(sigma, 4)
                        sigma_last = sigma
                        t_req_last = t_req
                        Q_sigma = Q_level * delta_q
                        c_update_res(p_self, out_types[5], 9, out_names[5])
                        c_update_res(p_self, out_types[6], sigma, out_names[6])
                        c_update_res(p_self, out_types[7], Q_sigma, out_names[7])
                        c_update_res(p_self, out_types[8], y_req, out_names[8])
```

```
                        file:write("   return: sigma: " .. sigma .. ", flux: " .. flux_last .. ",
Q: " ..Q_sigma .. ", err 9 \n")
                        file:write("integrator::run_logic() end \n\n")
                        return "double", sigma, "double",  flux_last, "double", Q_sigma, "int", 9
-----------------------------------------------------------------------------------------------
                end
        elseif req_type == 2 then
                if flux_last > 0 then
                        Q_level = Q_level + 1
                        sigma = delta_q / flux_last
                        sigma = round(sigma, 4)
                        sigma_last = sigma
                        t_req_last = t_req
                        y_req_last = 0
                        Q_sigma = Q_level * delta_q
                        c_update_res(p_self, out_types[5], 1, out_names[5])
                        c_update_res(p_self, out_types[6], sigma, out_names[6])
                        c_update_res(p_self, out_types[7], Q_sigma, out_names[7])
                        c_update_res(p_self, out_types[8], 0, out_names[8]) -- y_req

                        file:write("   return: sigma: " .. sigma .. ", flux: " .. flux_last .. ",
Q: " .. Q_sigma .. ", err 1 \n")
                        file:write("integrator::run_logic() end \n\n")
                        return "double", sigma, "double",  flux_last, "double", Q_sigma, "int", 1
                elseif flux_last < 0 then
                        Q_level = Q_level - 1
                        sigma = delta_q / flux_last * -1
                        sigma = round(sigma, 4)
                        sigma_last = sigma
                        t_req_last = t_req
                        y_req_last = delta_q
                        Q_sigma = Q_level * delta_q
                        c_update_res(p_self, out_types[5], 4, out_names[5])
                        c_update_res(p_self, out_types[6], sigma, out_names[6])
                        c_update_res(p_self, out_types[7], Q_sigma, out_names[7])
                        c_update_res(p_self, out_types[8], 0, out_names[8]) -- y_req

                        file:write("   return: sigma: " .. sigma .. ", flux: " .. flux_last .. ",
Q: " .. Q_sigma .. ", err 4 \n")
                        file:write("integrator::run_logic() end \n\n")
                        return "double", sigma, "double",  flux_last, "double", Q_sigma, "int", 4
                end
        elseif req_type == 3 then
                t_req_last = 0
                flux_last = 0
                Q_level = 0
                sigma_last = 0
                y_req_last = 0
                reset_count = reset_count - 1
                c_update_res(p_self, out_types[1], reset_count, out_names[1])
                c_update_res(p_self, out_types[2], reset_count, out_names[2])
                c_update_res(p_self, out_types[3], reset_count, out_names[3])
                c_update_res(p_self, out_types[4], reset_count, out_names[4])
                c_update_res(p_self, out_types[5], reset_count, out_names[5])
                c_update_res(p_self, out_types[6], reset_count, out_names[6])
                c_update_res(p_self, out_types[7], reset_count, out_names[7])
                file:write("integrator::run_logic() end \n\n")
                return
        end
end
--panaudojus sita funkcija, bus uzdarytas failas, todel skriptas feilins, jei bandys rasyti i
faila
function end_work()
        file:write("lua baige darba\n")
        file:close()
end
--funkcija klaidu aptikimui
--klaidos parametras visada grazinamas paskutinis pvz: x, y, ok = funkcija()
function kill(ok, err_msg, method, message)
        if ok == 0 then
```

```
                if err_msg == "" then
                else
                        file:write(err_msg .. "\n")
                end
                if methos == "" then
                else
                        file:write(method .." end\n")
                end
                if message == "" then
                else
                        file:write(message .. "\n")
                end
                return true
        end
        return false
end
--***PAGRINDINES FUNKCIJOS PABAIGA**************
--*****************************************
function round(number, digits)
    return tonumber(string.format("%." .. (digits or 0) .. "f", number))
end
```

7.2. Failas lua_sum_1.lua

```
--[[*************************************************************************
        ****ZEMIAU ESANCIO TEKSTO NEKEISTI*************************************]]--
function exists(n)
                local f = io.open(n)
                if f == nil then
                        return false
                else
                        f = io.close()
                        return true
                end
end
f_num = 0
t = true
f_name = os.time() .. "_lua_sum_1_log_" .. f_num .. ".txt"
while t do
        if exists(f_name) then
                f_num = f_num + 1
                f_name = os.time() .. "_lua_sum_1_log_" .. f_num .. ".txt"
        else
                t = false
        end
end

local file = io.open(f_name, "w")
file:write("lua pradejo darba\n")
--[[*************************************************************************
        ****ZEMIAU ESANTI TEKSTA GALIMA KEISTI**********************************]]--
local p_self

local in_names
local in_values = {}
local out_names
local out_values = {}
local beta_1
local beta_2
local beta_3
local koef_1
local koef_2
local koef_3
local koef_4
local koef_5
local x1
local x2
local x3
local reset_count
--*****************************************
```

```
--***HEAD PRADZIA*****************************
function head(pointer)
	file:write("lua_sum_1_log::head()\n")
	shape_type = {"circle"}
	in_types = {"double", "double", "double", "double", "double"}
	in_names = {"k12", "k13", "kel", "k21", "k31"}
	in_defaults = {1, 1, 1, 1, 1}
	in_controls = {"text_box", "text_box", "text_box", "text_box", "text_box"}
	out_types = {"double", "double", "double", "double", "double", "double", "double"}
	out_names = {"X1", "X2", "X3", "Y", "beta_1", "beta_2", "beta_3"}
	out_rep_types = {"chart", "chart", "chart", "chart", "chart", "chart", "chart", "chart"}
	c_set_shape_type(pointer, shape_type[1])
	c_set_in_types(pointer, in_types[1], in_types[2], in_types[3], in_types[4], in_types[5])
	c_set_in_names(pointer, in_names[1], in_names[2], in_names[3], in_names[4], in_names[5])
	c_set_in_defaults(pointer, in_defaults[1], in_defaults[2], in_defaults[3],
in_defaults[4], in_defaults[5])
	c_set_in_controls(pointer, in_controls[1], in_controls[2], in_controls[3],
in_controls[4], in_controls[5])
	c_set_out_types(pointer, out_types[1], out_types[2], out_types[3], out_types[4],
out_types[5], out_types[6], out_types[7])
	c_set_out_names(pointer, out_names[1], out_names[2], out_names[3], out_names[4],
out_names[5], out_names[6], out_names[7])
	c_set_out_rep_types(pointer, out_rep_types[1], out_rep_types[2], out_rep_types[3],
out_rep_types[4], out_rep_types[5], out_rep_types[6], out_rep_types[7])
	file:write("lua_sum_1_log::head() end\n\n")
end
--***HEAD PABAIGA******************************
--*********************************************
--*********************************************
--***PAGRINDINES FUNKCIJOS PRADZIA**************
function prepare(pointer)
	file:write("lua_sum_1_log::prepare()\n")
	p_self = pointer
	in_values[1], in_values[2], in_values[3], in_values[4], in_values[5], ok =
c_get_init_vals(p_self, 5)
	beta_1 = 0
	beta_2 = 0
	beta_3 = 0
	koef_1 = in_values[1]
	koef_2 = in_values[2]
	koef_3 = in_values[3]
	koef_4 = in_values[4]
	koef_5 = in_values[5]
	x1 = 0
	x2 = 0
	x3 = 0
	beta_1 = koef_1 * (-1) - koef_2 - koef_3
	beta_2 = koef_4
	beta_3 = koef_5
	c_update_res(p_self, out_types[5], beta_1, out_names[5])
	c_update_res(p_self, out_types[6], beta_2, out_names[6])
	c_update_res(p_self, out_types[7], beta_3, out_names[7])
	reset_count = 0
	file:write("lua_sum_1_log::prepare() end\n\n")
end

function run_logic(x_1, x_2, x_3, req_type)
	if req_type == 3 then
		file:write("lua_sum_1_log::run_logic(" .. x_1 .. " , " .. x_2 .. " , " .. x_3 ..
") \n")
		x1 = 0
		x2 = 0
		x3 = 0
		reset_count = reset_count - 1
		c_update_res(p_self, out_types[1], reset_count, out_names[1])
		c_update_res(p_self, out_types[2], reset_count, out_names[2])
		c_update_res(p_self, out_types[3], reset_count, out_names[3])
		c_update_res(p_self, out_types[4], reset_count, out_names[4])
		c_update_res(p_self, out_types[5], reset_count, out_names[5])
		c_update_res(p_self, out_types[6], reset_count, out_names[6])
		c_update_res(p_self, out_types[7], reset_count, out_names[7])
```

```
                file:write("return y = " .. y .. ", x1 = " .. x1 .. ", x2 = " .. x2 .. ", x3 = "
.. x3 .. "\n")
                file:write("lua_sum_1_log::run_logic() end\n\n")
                return
        end
        --atliekam logikos veiksmus
        file:write("lua_sum_1_log::run_logic(" .. x_1 .. " , " .. x_2 .. " , " .. x_3 .. ") \n")
        x1 = x_1
        x2 = x_2
        x3 = x_3
        c_update_res(p_self, out_types[1], x1, out_names[1])
        c_update_res(p_self, out_types[2], x2, out_names[2])
        c_update_res(p_self, out_types[3], x3, out_names[3])
        y = x1 * beta_1 + x2 * beta_2 + x3 * beta_3
        y = round(y, 4)
        c_update_res(p_self, out_types[4], y, out_names[4])
        file:write("return y = " .. y .. ", x1 = " .. x1 .. ", x2 = " .. x2 .. ", x3 = " .. x3 ..
"\n")
        file:write("lua_sum_1_log::run_logic() end\n\n")
        return "double", y, "double", x1, "double", x2, "double", x3
end

function round(number, digits)
    return tonumber(string.format("%." .. (digits or 0) .. "f", number))
end
--panaudojus sita funkcija, bus uzdarytas failas, todel skriptas feilins, jei bandys rasyti i
faila
function end_work()
        file:write("lua baige darba\n")
        file:close()
end

function kill(ok, err_msg, method, message)
        if ok == 0 then
                if err_msg == "" then
                else
                        file:write(err_msg .. "\n")
                end
                if methos == "" then
                else
                        file:write(method .." end\n")
                end
                if message == "" then
                else
                        file:write(message .. "\n")
                end
                return true
        end
        return false
end
--***PAGRINDINES FUNKCIJOS PABAIGA**************
--****************************************
```

7.3. Failas lua_sum_2.lua

```
--[[*************************************************************************
        ****ZEMIAU ESANCIO TEKSTO NEKEISTI*************************************]]--
function exists(n)
                local f = io.open(n)
                if f == nil then
                        return false
                else
                        f = io.close()
                        return true
                end
end
f_num = 0
t = true
f_name = os.time() .. "_lua_sum_2_log_" .. f_num .. ".txt"
while t do
```

```
        if exists(f_name) then
                f_num = f_num + 1
                f_name = os.time() .. "_lua_sum_2_log_" .. f_num .. ".txt"
        else
                t = false
        end
end
local file = io.open(f_name, "w")
file:write("lua pradejo darba\n")
--[[*********************************************************************
        ****ZEMIAU ESANTI TEKSTA GALIMA KEISTI**********************************]]--
local p_self
local in_names
local in_values = {}
local out_names
local out_values = {}
local beta_1
local beta_2
local koef_1
local koef_2
local x1
local x2
local reset_count
--*********************************************
--***HEAD PRADZIA*****************************
function head(pointer)
        file:write("lua_sum_2_log::head()\n")
        shape_type = {"circle"}
        in_types = {"double", "double"}
        in_names = {"k12", "k21"}
        in_defaults = {1, 1}
        in_controls = {"text_box", "text_box"}
        out_types = {"double", "double", "double", "double", "double"}
        out_names = {"X1", "X2", "Y", "beta_1", "beta_2"}
        out_rep_types = {"chart", "chart", "chart", "chart", "chart"}
        c_set_shape_type(pointer, shape_type[1])
        c_set_in_types(pointer, in_types[1], in_types[2])
        c_set_in_names(pointer, in_names[1], in_names[2])
        c_set_in_defaults(pointer, in_defaults[1], in_defaults[2])
        c_set_in_controls(pointer, in_controls[1], in_controls[2])
        c_set_out_types(pointer, out_types[1], out_types[2], out_types[3], out_types[4],
out_types[5])
        c_set_out_names(pointer, out_names[1], out_names[2], out_names[3], out_names[4],
out_names[5])
        c_set_out_rep_types(pointer, out_rep_types[1], out_rep_types[2], out_rep_types[3],
out_rep_types[4], out_rep_types[5])
        file:write("lua_sum_2_log::head() end\n\n")
end
--***HEAD PABAIGA****************************
--*********************************************
--*********************************************
--***PAGRINDINES FUNKCIJOS PRADZIA**************
function prepare(pointer)
        file:write("lua_sum_2_log::prepare()\n")
        --sudarom reikiamas strukturas, inicializuojam kintamuosius
        p_self = pointer
        in_values[1], in_values[2], ok = c_get_init_vals(p_self, 2)
        beta_1 = 0
        beta_2 = 0
        koef_1 = in_values[1]
        koef_2 = in_values[2]
        x1 = 0
        x2 = 0
        beta_1 = koef_1
        beta_2 = koef_2 * (-1)
        c_update_res(p_self, out_types[4], beta_1, out_names[4])
        c_update_res(p_self, out_types[5], beta_2, out_names[5])
        reset_count = 0
        file:write("lua_sum_2_log::prepare() end\n\n")
end
function run_logic(x_1, x_2, req_type)
```

```
        if req_type == 3 then
                file:write("lua_sum_2_log::run_logic(" .. x_1 .. " , " .. x_2 .. " , " .. x_3 ..
") \n")
                x1 = 0
                x2 = 0
                reset_count = reset_count - 1
                c_update_res(p_self, out_types[1], reset_count, out_names[1])
                c_update_res(p_self, out_types[2], reset_count, out_names[2])
                c_update_res(p_self, out_types[3], reset_count, out_names[3])
                c_update_res(p_self, out_types[4], reset_count, out_names[4])
                c_update_res(p_self, out_types[5], reset_count, out_names[5])
                file:write("return y = " .. y .. ", x1 = " .. x1 .. ", x2 = " .. x2 .. "\n")
                file:write("lua_sum_2_log::run_logic() end\n\n")
                return
        end
        file:write("lua_sum_2_log::run_logic(" .. x_1 .. " , " .. x_2 ..") \n")
        x1 = x_1
        x2 = x_2
        c_update_res(p_self, out_types[1], x1, out_names[1])
        c_update_res(p_self, out_types[2], x2, out_names[2])
        y = x1 * beta_1 + x2 * beta_2
        y = round(y, 4)
        c_update_res(p_self, out_types[3], y, out_names[3])
        file:write("return y = " .. y .. ", x1 = " .. x1 .. ", x2 = " .. x2 .. "\n")
        file:write("lua_sum_2_log::run_logic() end\n\n")
        return "double", y, "double", x1, "double", x2
end
function round(number, digits)
    return tonumber(string.format("%." .. (digits or 0) .. "f", number))
end
function end_work()
        file:write("lua baige darba\n")
        file:close()
end
function kill(ok, err_msg, method, message)
        if ok == 0 then
                if err_msg == "" then
                else
                        file:write(err_msg .. "\n")
                end
                if methos == "" then
                else
                        file:write(method .." end\n")
                end
                if message == "" then
                else
                        file:write(message .. "\n")
                end
                return true
        end
        return false
end
--***PAGRINDINES FUNKCIJOS PABAIGA**************
--*****************************************
```

7.4. Failas lua_sum_3.lua

```
--[[*************************************************************************
        ****ZEMIAU ESANCIO TEKSTO NEKEISTI*************************************]]--
function exists(n)
                local f = io.open(n)
                if f == nil then
                        return false
                else
                        f = io.close()
                        return true
                end
end
f_num = 0
t = true
```

```
f_name = os.time() .. "_lua_sum_3_log_" .. f_num .. ".txt"
while t do
	if exists(f_name) then
		f_num = f_num + 1
		f_name = os.time() .. "_lua_sum_3_log_" .. f_num .. ".txt"
	else
		t = false
	end
end
local file = io.open(f_name, "w")
file:write("lua pradejo darba\n")
--[[**************************************************************************
	****ZEMIAU ESANTI TEKSTA GALIMA KEISTI**********************************]]--
local p_self
local in_names
local in_values = {}
local out_names
local out_values = {}
local beta_1
local beta_2
local koef_1
local koef_2
local x1
local x2
local reset_count
--*******************************************
--***HEAD PRADZIA*****************************
function head(pointer)
	file:write("lua_sum_3_log::head()\n")
	shape_type = {"circle"}
	in_types = {"double", "double"}
	in_names = {"k13", "k31"}
	in_defaults = {1, 1}
	in_controls = {"text_box", "text_box"}
	out_types = {"double", "double", "double", "double", "double"}
	out_names = {"X1", "X2", "Y", "beta_1", "beta_2"}
	out_rep_types = {"chart", "chart", "chart", "chart", "chart"}
	c_set_shape_type(pointer, shape_type[1])
	c_set_in_types(pointer, in_types[1], in_types[2])
	c_set_in_names(pointer, in_names[1], in_names[2])
	c_set_in_defaults(pointer, in_defaults[1], in_defaults[2])
	c_set_in_controls(pointer, in_controls[1], in_controls[2])
	c_set_out_types(pointer, out_types[1], out_types[2], out_types[3], out_types[4],
out_types[5])
	c_set_out_names(pointer, out_names[1], out_names[2], out_names[3], out_names[4],
out_names[5])
	c_set_out_rep_types(pointer, out_rep_types[1], out_rep_types[2], out_rep_types[3],
out_rep_types[4], out_rep_types[5])
	file:write("lua_sum_3_log::head() end\n\n")
end
--***HEAD PABAIGA*****************************
--*******************************************
--*******************************************
--***PAGRINDINES FUNKCIJOS PRADZIA**************
function prepare(pointer)
	file:write("lua_sum_3_log::prepare()\n")
	--sudarom reikiamas strukturas, inicializuojam kintamuosius
	p_self = pointer
	in_values[1], in_values[2], ok = c_get_init_vals(p_self, 2)
	beta_1 = 0
	beta_2 = 0
	koef_1 = in_values[1]
	koef_2 = in_values[2]
	x1 = 0
	x2 = 0
	beta_1 = koef_1
	beta_2 = koef_2 * (-1)
	c_update_res(p_self, out_types[4], beta_1, out_names[4])
	c_update_res(p_self, out_types[5], beta_2, out_names[5])
	reset_count = 0
	file:write("lua_sum_3_log::prepare() end\n\n")
```

```
end
function run_logic(x_1, x_2, req_type)
	if req_type == 3 then
		file:write("lua_sum_3_log::run_logic(" .. x_1 .. " , " .. x_2 .. " , " .. x_3 ..
") \n")
		x1 = 0
		x2 = 0
		reset_count = reset_count - 1
		c_update_res(p_self, out_types[1], reset_count, out_names[1])
		c_update_res(p_self, out_types[2], reset_count, out_names[2])
		c_update_res(p_self, out_types[3], reset_count, out_names[3])
		c_update_res(p_self, out_types[4], reset_count, out_names[4])
		c_update_res(p_self, out_types[5], reset_count, out_names[5])
		file:write("return y = " .. y .. ", x1 = " .. x1 .. ", x2 = " .. x2 .. "\n")
		file:write("lua_sum_3_log::run_logic() end\n\n")
		return
	end
	file:write("lua_sum_3_log::run_logic(" .. x_1 .. " , " .. x_2 ..") \n")
	x1 = x_1
	x2 = x_2
	c_update_res(p_self, out_types[1], x1, out_names[1])
	c_update_res(p_self, out_types[2], x2, out_names[2])
	y = x1 * beta_1 + x2 * beta_2
	c_update_res(p_self, out_types[3], y, out_names[3])
	y = round(y, 4)
	file:write("return y = " .. y .. ", x1 = " .. x1 .. ", x2 = " .. x2 .. "\n")
	file:write("lua_sum_3_log::run_logic() end\n\n")
	return "double", y, "double", x1, "double", x2
end
function round(number, digits)
    return tonumber(string.format("%." .. (digits or 0) .. "f", number))
end
function end_work()
	file:write("lua baige darba\n")
	file:close()
end
function kill(ok, err_msg, method, message)
	if ok == 0 then
		if err_msg == "" then
		else
			file:write(err_msg .. "\n")
		end
		if methos == "" then
		else
			file:write(method .." end\n")
		end
		if message == "" then
		else
			file:write(message .. "\n")
		end
		return true
	end
	return false
end
--***PAGRINDINES FUNKCIJOS PABAIGA**************
--*****************************************
```

7.5. Failas lua_pump.lua

```
--[[*********************************************************************
	****ZEMIAU ESANCIO TEKSTO NEKEISTI*************************************]]--
function exists(n)
		local f = io.open(n)
		if f == nil then
			return false
		else
			f = io.close()
			return true
		end
end
```

```
f_num = 0
t = true
f_name = os.time() .. "_lua_pump_log_" .. f_num .. ".txt"
while t do
	if exists(f_name) then
		f_num = f_num + 1
		f_name = os.time() .. "_lua_pump_log_" .. f_num .. ".txt"
	else
		t = false
	end
end
local file = io.open(f_name, "w")
file:write("lua pradejo darba\n")
--[[**************************************************************************
	****ZEMIAU ESANTI TEKSTA GALIMA KEISTI**********************************]]--
local p_self
local in_names
local in_values = {}
local out_names
local out_values = {}
local t_req
local t_deliv
local state
local free_time
local t_reject
local deliv_ratio
local dose_count
local reject_count
local active_time
local reset_count
local arr_t_req = {}
local arr_t_deliv = {}
local arr_t_reject = {}
local arr_free_time = {}
local arr_active_time = {}
local arr_wait_time = {}
local arr_state = {}
--*****************************************
--***HEAD PRADZIA*****************************
function head(pointer)
	file:write("lua_pump_log::head()\n")
	shape_type = {"circle"}
	in_types = {"double"}
	in_names = {"T_minuciu"}
	in_defaults = {10}
	in_controls = {"text_box"}
	out_types = {"double", "double", "int", "double", "double", "double", "double", "int",
"int", "double", "double"}
	out_names = {"t_req", "t_deliv", "state", "free_time", "wait_time", "t_reject",
"deliv_ratio", "dose_count", "reject_count", "ave_free_time", "ave_wait_time"}
	out_rep_types = {"chart", "chart", "chart", "chart", "chart", "cahrt", "chart", "chart",
"chart", "chart", "chart"}
	c_set_shape_type(pointer, shape_type[1])
	c_set_in_types(pointer, in_types[1])
	c_set_in_names(pointer, in_names[1])
	c_set_in_defaults(pointer, in_defaults[1])
	c_set_in_controls(pointer, in_controls[1])
	c_set_out_types(pointer, out_types[1], out_types[2], out_types[3], out_types[4],
out_types[5], out_types[6], out_types[7], out_types[8], out_types[9], out_types[10],
out_types[11])
	c_set_out_names(pointer, out_names[1], out_names[2], out_names[3], out_names[4],
out_names[5], out_names[6], out_names[7], out_names[8], out_names[9], out_names[10],
out_names[11])
	c_set_out_rep_types(pointer, out_rep_types[1], out_rep_types[2], out_rep_types[3],
out_rep_types[4], out_rep_types[5], out_rep_types[6], out_rep_types[7], out_rep_types[8],
out_rep_types[9], out_rep_types[10], out_rep_types[11])
	file:write("lua_pump_log::head() end\n\n")
end
--***HEAD PABAIGA*****************************
--*****************************************
--*****************************************
```

```
--***PAGRINDINES FUNKCIJOS PRADZIA**************
function prepare(pointer)
	file:write("lua_pump_log::prepare()\n")
	p_self = pointer
	in_values[1], ok = c_get_init_vals(p_self, 1)
	t_req = 0
	t_deliv = 0
	state = 0 -- 0 - laukia aktyvavimo, 1 - aktyvi
	free_time = 0
	t_reject = 0
	deliv_ratio = 0
	dose_count = 0
	reject_count = 0
	active_time = 0
	active_time = in_values[1]
	reset_count = 0
	arr_free_time["ave"] = 0
	arr_wait_time["ave"] = 0
	arr_t_req["count"] = 0
	arr_t_deliv["count"] = 0
	arr_t_reject["count"] = 0
	arr_free_time["count"] = 0
	arr_wait_time["count"] = 0
	arr_state["count"] = 0
	arr_t_req[0] = 0
	arr_t_deliv[0] = 0
	arr_t_reject[0] = 0
	arr_free_time[0] = 0
	arr_wait_time[0] = 0
	arr_state[0] = 0
	file:write("lua_pump_log::prepare() end\n\n")
end
function run_logic(logic_num, T)
	file:write("lua_pump::run_logic(" .. logic_num .. " , " .. T .. ") \n")
	if logic_num == 0 then --skaiciuojam laikus
		t_req = T
		state_r = 0
		arr_t_req["count"] = arr_t_req["count"] + 1
		arr_t_req[arr_t_req["count"]] = t_req
		if t_req >= t_deliv then
			state = 0
			arr_state["count"] = arr_state["count"] + 1
			arr_state[arr_state["count"]] = state
			state = 1
			state_r = 0
			free_time = t_req - t_deliv
			arr_free_time["count"] = arr_free_time["count"] + 1
			arr_free_time[arr_free_time["count"]] = free_time
			t_deliv = t_req + active_time
			arr_t_deliv["count"] = arr_t_deliv["count"] + 1
			arr_t_deliv[arr_t_deliv["count"]] = t_deliv
		else
			arr_state["count"] = arr_state["count"] + 1
			arr_state[arr_state["count"]] = state
			t_reject = t_req
			arr_t_reject["count"] = arr_t_reject["count"] + 1
			arr_t_reject[arr_t_reject["count"]] = t_reject
			wait_time = t_deliv - t_reject
			arr_wait_time["count"] = arr_wait_time["count"] + 1
			arr_wait_time[arr_wait_time["count"]] = wait_time
			state_r = 1
		end
		file:write("arr_t_req["..arr_t_req["count"].."] = " ..
arr_t_req[arr_t_req["count"]] .. " | arr_state["..arr_state["count"].."] = " ..
arr_state[arr_state["count"]] .. " | arr_free_time["..arr_free_time["count"].."] = " ..
arr_free_time[arr_free_time["count"]] .. " | arr_t_deliv["..arr_t_deliv["count"].."] = " ..
arr_t_deliv[arr_t_deliv["count"]] .. " | arr_t_reject["..arr_t_reject["count"].."] = " ..
arr_t_reject[arr_t_reject["count"]] .. " | arr_wait_time["..arr_wait_time["count"].."] = " ..
arr_wait_time[arr_wait_time["count"]] .. "\n")
		file:write("return " .. t_deliv .. "\n")
		file:write("lua_pump::run_logic() end\n")
```

```
        return "double", t_deliv, "int", state_r
elseif logic_num == 1 then --skaiciuojam statistikas
        deliv_ratio = arr_t_deliv["count"]/arr_t_req["count"]
        file:write("deliv_ratio = " .. deliv_ratio .. "\n")
        temp = 0
        for i = 1, arr_free_time["count"] do
                temp = temp + arr_free_time[i]
        end
        temp = temp/arr_free_time["count"]
        arr_free_time["ave"] = temp
        file:write("arr_free_time[ave] = " .. arr_free_time["ave"] .. "\n")
        temp = 0
        for i = 1, arr_wait_time["count"] do
                temp = temp + arr_wait_time[i]
        end
        temp = temp/arr_wait_time["count"]
        arr_wait_time["ave"] = temp
        file:write("arr_wait_time[ave] = " .. arr_wait_time["ave"] .. "\n")
        file:write("return " .. arr_t_req["count"] .. ", " .. deliv_ratio .. "\n")
        file:write("file_logic::run_logic() end\n\n")
        return "int", arr_t_req["count"], "double", deliv_ratio
elseif logic_num == 2 then --updeitinam reiksmes
        for i = 1, arr_t_req["count"] do
                c_update_res(p_self, out_types[1], arr_t_req[i], out_names[1])
        end
        for i = 1, arr_t_deliv["count"] do
                c_update_res(p_self, out_types[2], arr_t_deliv[i], out_names[2])
        end
        for i = 1, arr_t_reject["count"] do
                c_update_res(p_self, out_types[6], arr_t_reject[i], out_names[6])
        end
        for i = 1, arr_free_time["count"] do
                c_update_res(p_self, out_types[4], arr_free_time[i], out_names[4])
        end
        c_update_res(p_self, out_types[10], arr_free_time["ave"], out_names[10])
        for i = 1, arr_wait_time["count"] do
                c_update_res(p_self, out_types[5], arr_wait_time[i], out_names[5])
        end
        c_update_res(p_self, out_types[11], arr_wait_time["ave"], out_names[11])
        for i = 1, arr_state["count"] do
                c_update_res(p_self, out_types[3], arr_state[i], out_names[3])
        end
        c_update_res(p_self, out_types[7], deliv_ratio, out_names[7])
        c_update_res(p_self, out_types[8], arr_t_deliv["count"], out_names[8])
        c_update_res(p_self, out_types[9], arr_t_reject["count"], out_names[9])
        file:write("lua_pump::run_logic() end\n\n")
        return
elseif logic_num == 3 then --resetinam reiksmes
        reset_count = reset_count - 1
        c_update_res(p_self, out_types[1], reset_count, out_names[1])
        c_update_res(p_self, out_types[2], reset_count, out_names[2])
        c_update_res(p_self, out_types[3], reset_count, out_names[3])
        c_update_res(p_self, out_types[4], reset_count, out_names[4])
        c_update_res(p_self, out_types[5], reset_count, out_names[5])
        c_update_res(p_self, out_types[6], reset_count, out_names[6])
        t_req = 0
        t_deliv = 0
        state = 0 -- 0 - laukia aktyvavimo, 1 - aktyvi
        free_time = 0
        t_reject = 0
        deliv_ratio = 0
        dose_count = 0
        reject_count = 0
        arr_free_time["ave"] = 0
        arr_wait_time["ave"] = 0
        arr_t_req["count"] = 0
        arr_t_deliv["count"] = 0
        arr_t_reject["count"] = 0
        arr_free_time["count"] = 0
        arr_wait_time["count"] = 0
        arr_state["count"] = 0
```

```
                arr_t_req[0] = 0
                arr_t_deliv[0] = 0
                arr_t_reject[0] = 0
                arr_free_time[0] = 0
                arr_wait_time[0] = 0
                arr_state[0] = 0
                file:write("lua_pump::run_logic() end\n\n")
                return
        end
end
function end_work()
        file:write("lua baige darba\n")
        file:close()
end
function kill(ok, err_msg, method, message)
        if ok == 0 then
                if err_msg == "" then
                else
                        file:write(err_msg .. "\n")
                end
                if methos == "" then
                else
                        file:write(method .." end\n")
                end
                if message == "" then
                else
                        file:write(message .. "\n")
                end
                return true
        end
        return false
end
--***PAGRINDINES FUNKCIJOS PABAIGA**************
--****************************************
```

7.6. Failas lua_pumpi.lua

```
--[[*************************************************************************
        ****ZEMIAU ESANCIO TEKSTO NEKEISTI*************************************]]--
function exists(n)
                local f = io.open(n)
                if f == nil then
                        return false
                else
                        f = io.close()
                        return true
                end
end
f_num = 0
t = true
f_name = os.time() .. "_lua_pump_i_log" .. f_num .. ".txt"
while t do
        if exists(f_name) then
                f_num = f_num + 1
                f_name = os.time() .. "_lua_pump_i_log" .. f_num .. ".txt"
        else
                t = false
        end
end
local file = io.open(f_name, "w")
file:write("lua pradejo darba\n")
--[[*********************************************************************
        ****ZEMIAU ESANTI TEKSTA GALIMA KEISTI**********************************]]--
local p_self
local in_names
local in_values = {}
local out_names
local out_values = {}
local state
local t_deliv
```

```
local t_req
local t_req_l
local arr_t_req = {}
local arr_state = {}
local arr_t_pred_deliv = {}
local arr_t_deliv = {}
local arr_deliv_time = {}
local arr_free_time = {}
local arr_dose_count = {}
local ave_free_time
local ave_deliv_time
local active_time
local reset_count
--*****************************************
--***HEAD PRADZIA*********************************
function head(pointer)
	file:write("lua_pump_i_log::head()\n")
	shape_type = {"circle"}
	in_types = {"double"}
	in_names = {"T_minuciu"}
	in_defaults = {10}
	in_controls = {"text_box"}
	out_types = {"double", "int", "double", "double", "double", "double", "int", "int",
"double", "double" }
	out_names = {"t_req", "state", "t_pred_deliv", "t_deliv", "deliv_time", "free_time",
"dose_count", "req_count", "ave_free_time", "ave_deliv_time" }
	out_rep_types = {"chart", "chart", "chart", "chart", "chart", "chart", "chart", "chart",
"chart", "chart"}
	c_set_shape_type(pointer, shape_type[1])
	c_set_in_types(pointer, in_types[1])
	c_set_in_names(pointer, in_names[1])
	c_set_in_defaults(pointer, in_defaults[1])
	c_set_in_controls(pointer, in_controls[1])
	c_set_out_types(pointer, out_types[1], out_types[2], out_types[3], out_types[4],
out_types[5], out_types[6], out_types[7], out_types[8], out_types[9], out_types[10])
	c_set_out_names(pointer, out_names[1], out_names[2], out_names[3], out_names[4],
out_names[5], out_names[6], out_names[7], out_names[8], out_names[9], out_names[10])
	c_set_out_rep_types(pointer, out_rep_types[1], out_rep_types[2], out_rep_types[3],
out_rep_types[4], out_rep_types[5], out_rep_types[6], out_rep_types[7], out_rep_types[8],
out_rep_types[9], out_rep_types[10])
	file:write("lua_pump_i_log::head() end\n\n")
end
--***HEAD PABAIGA*********************************
--*****************************************
--*****************************************
--***PAGRINDINES FUNKCIJOS PRADZIA**************
function prepare(pointer)
	file:write("lua_pump_i_log::prepare()\n")
	p_self = pointer
	in_values[1], ok = c_get_init_vals(p_self, 1)
	active_time = 0
	active_time = in_values[1]
	state = 0
	t_deliv = 0
	t_req = 0
	t_req_l = 0
	arr_t_req["count"] = 0
	arr_state["count"] = 0
	arr_t_pred_deliv["count"] = 0
	arr_t_deliv["count"] = 0
	arr_deliv_time["count"] = 0
	arr_free_time["count"] = 0
	ave_free_time = 0
	ave_deliv_time = 0
	arr_t_req[0] = 0
	arr_state[0] = 0
	arr_t_pred_deliv[0] = 0
	arr_t_deliv[0] = 0
	arr_deliv_time[0] = 0
	arr_free_time[0] = 0
	reset_count = 0
```

```
	file:write("lua_pump_i_log::prepare() end\n\n")
end
function run_logic(logic_num, T)
	file:write("lua_pump_i_log::run_logic(" .. logic_num .. " , " .. T .. ") \n")
	if logic_num == 0 then --skaiciuojam laikus
		t_req = T
		arr_t_req["count"] = arr_t_req["count"] + 1
		arr_t_req[arr_t_req["count"]] = t_req
		if t_req >= t_deliv then
			state = 0
			arr_state["count"] = arr_state["count"] + 1
			arr_state[arr_state["count"]] = state
			state = 1
			free_time = t_req - t_deliv
			arr_free_time["count"] = arr_free_time["count"] + 1
			arr_free_time[arr_free_time["count"]] = free_time
			if t_deliv > 0 then
				arr_t_deliv["count"] = arr_t_deliv["count"] + 1
				arr_t_deliv[arr_t_deliv["count"]] = t_deliv
				arr_deliv_time["count"] = arr_deliv_time["count"] + 1
				arr_deliv_time[arr_deliv_time["count"]] = t_deliv - t_req_l
			end
			t_req_l = t_req
			t_deliv = t_req_l + active_time
			arr_t_pred_deliv["count"] = arr_t_pred_deliv["count"] + 1
			arr_t_pred_deliv[arr_t_pred_deliv["count"]] = t_deliv
			pred_deliv_time = t_deliv
		else
			arr_state["count"] = arr_state["count"] + 1
			arr_state[arr_state["count"]] = state
			t_deliv = t_req + active_time
			arr_t_pred_deliv["count"] = arr_t_pred_deliv["count"] + 1
			arr_t_pred_deliv[arr_t_pred_deliv["count"]] = t_deliv
			pred_deliv_time = t_deliv
		end
		file:write("arr_t_req["..arr_t_req["count"].."] = " ..
arr_t_req[arr_t_req["count"]] .. " | arr_state["..arr_state["count"].."] = " ..
arr_state[arr_state["count"]] .. " | arr_free_time["..arr_free_time["count"].."] = " ..
arr_free_time[arr_free_time["count"]] .. " | arr_t_deliv["..arr_t_deliv["count"].."] = " ..
arr_t_deliv[arr_t_deliv["count"]] .. " | arr_t_pred_deliv["..arr_t_pred_deliv["count"].."] = " ..
arr_t_pred_deliv[arr_t_pred_deliv["count"]] .. " | arr_deliv_time["..arr_deliv_time["count"].."]
= " .. arr_deliv_time[arr_deliv_time["count"]] .. "\n")
		file:write("return " .. pred_deliv_time .. "\n")
		file:write("lua_pump_i_log::run_logic() end\n\n")
		return "double", pred_deliv_time , "int", arr_state[arr_state["count"]]--galime
grazinti tik numatoma pristatymo laika, nes yra galimybe pristatymo pratesimui
	elseif logic_num == 1 then --skaiciuojam statistikas
		if t_deliv > 0 then
			arr_t_deliv["count"] = arr_t_deliv["count"] + 1
			arr_t_deliv[arr_t_deliv["count"]] = t_deliv
			arr_deliv_time["count"] = arr_deliv_time["count"] + 1
			arr_deliv_time[arr_deliv_time["count"]] = t_deliv - t_req_l
		end
		temp = 0
		for i = 1, arr_free_time["count"] do
			temp = temp + arr_free_time[i]
		end
		temp = temp/arr_free_time["count"]
		ave_free_time = temp
		temp = 0
		for i = 1, arr_deliv_time["count"] do
			temp = temp + arr_deliv_time[i]
		end
				temp = temp/arr_deliv_time["count"]
		ave_deliv_time = temp
		file:write("return " .. arr_t_req["count"] .. "," .. arr_t_deliv["count"] .. ","
.. ave_free_time .. "," .. ave_deliv_time.. "\n")
		file:write("lua_pump_i_log::run_logic() end\n\n")
		return "int", arr_t_req["count"], "int", arr_t_deliv["count"], "double",
ave_free_time, "double", ave_deliv_time
	elseif logic_num == 2 then --updeitinam reiksmes
```

```
                for i = 1, arr_t_req["count"] do
                        c_update_res(p_self, out_types[1], arr_t_req[i], out_names[1])
                end
                for i = 1, arr_state["count"] do
                        c_update_res(p_self, out_types[2], arr_state[i], out_names[2])
                end
                for i = 1, arr_t_pred_deliv["count"] do
                        c_update_res(p_self, out_types[3], arr_t_pred_deliv[i], out_names[3])
                end
                for i = 1, arr_t_deliv["count"] do
                        c_update_res(p_self, out_types[4], arr_t_deliv[i], out_names[4])
                end
                for i = 1, arr_deliv_time["count"] do
                        c_update_res(p_self, out_types[5], arr_deliv_time[i], out_names[5])
                end
                for i = 1, arr_free_time["count"] do
                        c_update_res(p_self, out_types[6], arr_free_time[i], out_names[6])
                end
                c_update_res(p_self, out_types[7], arr_deliv_time["count"], out_names[7])
                c_update_res(p_self, out_types[8], arr_t_req["count"], out_names[8])
                c_update_res(p_self, out_types[9], ave_free_time, out_names[9])
                c_update_res(p_self, out_types[10], ave_deliv_time, out_names[10])
                file:write("lua_pump_i_log::run_logic() end\n\n")
                --doziu kiekis, prasymu kiekis, vidutinis laisvas laikas, vidutinis dozes
pristatymo laikas
                return  "int", arr_t_req["count"], "int", arr_deliv_time["count"], "double",
ave_free_time, "double", ave_deliv_time
        elseif logic_num == 3 then --resetinam reiksmes
                reset_count = reset_count - 1
                c_update_res(p_self, out_types[1], reset_count, out_names[1])
                c_update_res(p_self, out_types[2], reset_count, out_names[2])
                c_update_res(p_self, out_types[3], reset_count, out_names[3])
                c_update_res(p_self, out_types[4], reset_count, out_names[4])
                c_update_res(p_self, out_types[5], reset_count, out_names[5])
                c_update_res(p_self, out_types[6], reset_count, out_names[6])
                state = 0
                t_deliv = 0
                t_req = 0
                t_req_l = 0
                arr_t_req["count"] = 0
                arr_state["count"] = 0
                arr_t_pred_deliv["count"] = 0
                arr_t_deliv["count"] = 0
                arr_deliv_time["count"] = 0
                arr_free_time["count"] = 0
                ave_free_time = 0
                ave_deliv_time = 0
                arr_t_req[0] = 0
                arr_state[0] = 0
                arr_t_pred_deliv[0] = 0
                arr_t_deliv[0] = 0
                arr_deliv_time[0] = 0
                arr_free_time[0] = 0
                file:write("lua_pump_i_log::run_logic() end\n\n")
                return
        end
end
function end_work()
        file:write("lua baige darba\n")
        file:close()
end
function kill(ok, err_msg, method, message)
        if ok == 0 then
                if err_msg == "" then
                else
                        file:write(err_msg .. "\n")
                end
                if methos == "" then
                else
                        file:write(method .." end\n")
                end
```

```
                if message == "" then
                else
                        file:write(message .. "\n")
                end
                return true
        end
        return false
end
--***PAGRINDINES FUNKCIJOS PABAIGA**************
--*****************************************
```

7.7. Failas lua_pain_A.lua

```
--[[*************************************************************************
        ****ZEMIAU ESANCIO TEKSTO NEKEISTI*************************************]]--
function exists(n)
                local f = io.open(n)
                if f == nil then
                        return false
                else
                        f = io.close()
                        return true
                end
end
f_num = 0
t = true
f_name = os.time() .. "_lua_pain_A_log_" .. f_num .. ".txt"
while t do
        if exists(f_name) then
                f_num = f_num + 1
                f_name = os.time() .. "_lua_pain_A_log_" .. f_num .. ".txt"
        else
                t = false
        end
end
local file = io.open(f_name, "w")
file:write("lua pradejo darba\n")
--[[*************************************************************************
        ****ZEMIAU ESANTI TEKSTA GALIMA KEISTI**********************************]]--
local p_self
local in_names
local in_values = {}
local out_names
local out_values = {}
local lamda
local random_seed
--*****************************************
--***HEAD PRADZIA*******************************
function head(pointer)
        file:write("file_pain_A::head()\n")
        shape_type = {"circle"}
        in_types = {"int"}
        in_names = {"lamda"}
        in_defaults = {5}
        in_controls = {"text_box"}
        out_types = {"double"}
        out_names = {"t_m"}
        out_rep_types = {"chart"}
        c_set_shape_type(pointer, shape_type[1])
        c_set_in_types(pointer, in_types[1])
        c_set_in_names(pointer, in_names[1])
        c_set_in_defaults(pointer, in_defaults[1])
        c_set_in_controls(pointer, in_controls[1])
        c_set_out_types(pointer, out_types[1])
        c_set_out_names(pointer, out_names[1])
        c_set_out_rep_types(pointer, out_rep_types[1])
        file:write("file_pain_A::head() end\n")
end
--***HEAD PABAIGA*****************************
--*****************************************
```

```
--*******************************************
--***PAGRINDINES FUNKCIJOS PRADZIA**************
function prepare(pointer)
	file:write("file_pain_A::prepare()\n")
	--sudarom reikiamas strukturas, inicializuojam kintamuosius
	p_self = pointer
	random_seed = os.time()
	file:write("random_seed = " .. random_seed .. "\n")
	in_values[1], ok = c_get_init_vals(p_self, 1)
	lamda = in_values[1]
	file:write("file_pain_A::prepare() end\n")
end
function run_logic(x)
	file:write("file_pain_A::run_logic()\n")
	math.randomseed(random_seed)
	rand_s = math.random(1, 50000)
	random_seed = random_seed + rand_s
	math.randomseed(random_seed)
	rand_s = math.random(1, 50000)
	random_seed = random_seed + rand_s
	math.randomseed(random_seed)
	rand = math.random()
	rand = math.random()
	rand = math.random()
	ln_ = math.log(rand)
	t_m = - 1 * (1 / lamda) * ln_
	file:write("random seed = " .. random_seed .. "\n")
	file:write("random        = " .. rand .. "\n")
	file:write("ln(rand)    = " .. ln_ .. "\n")
	file:write("t_m         = " .. t_m .. "\n")
	file:write("file_pain_A::run_logic() end\n")
	c_update_res(p_self, out_types[1], t_m, out_names[1])
	return "double", t_m --grazina delta_t
end
function end_work()
	file:write("lua baige darba\n")
	file:close()
end
function kill(ok, err_msg, method, message)
	if ok == 0 then
		if err_msg == "" then
		else
			file:write(err_msg .. "\n")
		end
		if methos == "" then
		else
			file:write(method .." end\n")
		end
		if message == "" then
		else
			file:write(message .. "\n")
		end
		return true
	end
	return false
end
--***PAGRINDINES FUNKCIJOS PABAIGA**************
--*******************************************
```

7.8. Failas lua_pain_B.lua

```
--[[*********************************************************************
	****ZEMIAU ESANCIO TEKSTO NEKEISTI*************************************]]--
function exists(n)
		local f = io.open(n)
		if f == nil then
			return false
		else
			f = io.close()
			return true
```

```
		end
end
f_num = 0
t = true
f_name = os.time() .. "_lua_pain_B_log_" .. f_num .. ".txt"
while t do
	if exists(f_name) then
		f_num = f_num + 1
		f_name = os.time() .. "_lua_pain_B_log_" .. f_num .. ".txt"
	else
		t = false
	end
end
local file = io.open(f_name, "w")
file:write("lua pradejo darba\n")
--[[*********************************************************************
	****ZEMIAU ESANTI TEKSTA GALIMA KEISTI**********************************]]--
local p_self
local in_names
local in_values = {}
local out_names
local out_values = {}
local q_min
local random_seed
local wait_min
local wait_max
--*******************************************
--***HEAD PRADZIA*****************************
function head(pointer)
	file:write("file_pain_B::head()\n")
	shape_type = {"circle"}
	in_types = {"double", "double", "double"}
	in_names = {"Q_level_min", "wait_min", "wait_max"}
	in_defaults = {70, 1, 5}
	in_controls = {"text_box", "text_box", "text_box"}
	out_types = {"double"}
	out_names = {"t_m"}
	out_rep_types = {"chart"}
	c_set_shape_type(pointer, shape_type[1])
	c_set_in_types(pointer, in_types[1], in_types[2], in_types[3])
	c_set_in_names(pointer, in_names[1], in_names[2], in_names[3])
	c_set_in_defaults(pointer, in_defaults[1], in_defaults[2], in_defaults[3])
	c_set_in_controls(pointer, in_controls[1], in_controls[2], in_controls[3])
	c_set_out_types(pointer, out_types[1])
	c_set_out_names(pointer, out_names[1])
	c_set_out_rep_types(pointer, out_rep_types[1])
	file:write("file_pain_B::head() end\n")
end
--***HEAD PABAIGA****************************
--*******************************************
--*******************************************
--***PAGRINDINES FUNKCIJOS PRADZIA**************
function prepare(pointer)
	file:write("file_pain_B::prepare()\n")
	--sudarom reikiamas strukturas, inicializuojam kintamuosius
	p_self = pointer
	random_seed = os.time()
	file:write("random_seed = " .. random_seed .. "\n")
	in_values[1], in_values[2], in_values[3], ok = c_get_init_vals(p_self, 3)
	q_min = in_values[1]
	wait_min = in_values[2]
	wait_max = in_values[3]
	file:write("file_pain_B::prepare() end\n")
end
function run_logic(q_level)
	file:write("file_pain_B::run_logic()\n")
	if q_level < q_min then
	math.randomseed(random_seed)
	rand_s = math.random(1, 50000)
	random_seed = random_seed + rand_s
	math.randomseed(random_seed)
```

```
        rand_s = math.random(1, 50000)
        random_seed = random_seed + rand_s
        math.randomseed(random_seed)
        rand = math.random()
        rand = math.random()
        rand = math.random()
        t_m = rand * (wait_max - wait_min) + wait_min
        file:write("QLevel            = " .. q_level .. "\n")
        file:write("random seed = " .. random_seed .. "\n")
        file:write("random         = " .. rand .. "\n")
        file:write("t_m          = " .. t_m .. "\n")
        file:write("file_pain_B::run_logic() end\n\n")
        c_update_res(p_self, out_types[1], t_m, out_names[1])
        return "double", t_m --grazina delta_t
        else
                t_m = -1
                c_update_res(p_self, out_types[1], t_m, out_names[1])
                return "double", t_m --grazina delta_t
        end
end
function end_work()
        file:write("lua baige darba\n")
        file:close()
end
function kill(ok, err_msg, method, message)
        if ok == 0 then
                if err_msg == "" then
                else
                        file:write(err_msg .. "\n")
                end
                if methos == "" then
                else
                        file:write(method .." end\n")
                end
                if message == "" then
                else
                        file:write(message .. "\n")
                end
                return true
        end
        return false
end
--***PAGRINDINES FUNKCIJOS PABAIGA**************
--*****************************************
```

7.9. Failas lua_logic_A.lua

```
--[[*************************************************************************
        ****ZEMIAU ESANCIO TEKSTO NEKEISTI*************************************]]--
function exists(n)
                local f = io.open(n)
                if f == nil then
                        return false
                else
                        f = io.close()
                        return true
                end
end
f_num = 0
t = true
f_name = os.time() .. "_lua_logic_A_log_" .. f_num .. ".txt"
while t do
        if exists(f_name) then
                f_num = f_num + 1
                f_name = os.time() .. "_lua_logic_A_log_" .. f_num .. ".txt"
        else
                t = false
        end
end
local file = io.open(f_name, "w")
```

```
file:write("lua pradejo darba\n")
--[[*****************************************************************************
       ****ZEMIAU ESANTI TEKSTA GALIMA KEISTI**********************************]]--
local p_self
local p_v_str
local in_names
local in_values = {}
local out_names
local out_values = {}
local graph_size
local graph_pointers = {}
local adj_list = {}
local shift_list = {}
local pain                      -- 1
local pump                      -- 2
local pump_i           -- 9
local integ1           -- 3
local integ2           -- 4
local integ3           -- 5
local integ1_i                 --10
local integ2_i         --11
local integ3_i         --12
local sum1                      -- 6
local sum2                      -- 7
local sum3                      -- 8
local sum1_i           --13
local sum2_i           --14
local sum3_i           --15
local Sim_time_h
local dose_size
local dose_parts
local compart_volume

--*********************************************
--***HEAD PRADZIA*****************************
function head(pointer)                                          --pointer - rodykle i c_object tipo
c++ objekta
       file:write("file_logic_A::head()\n")
       shape_type = {"logic"}
       in_types = {"double", "double", "double", "double"}
       in_names = {"Sim_time_h", "Dose_size", "Dose_parts", "Compart_volume"}
       in_defaults = {12, 1000, 10, 13}
       in_controls = {"text_box", "text_box", "text_box", "text_box"}
       out_types = {"double", "double", "double", "double", "double", "double", "double",
"double",
                             "double", "double", "double", "double", "double", "double",
"double", "double"}
       out_names = {"t_req"  , "tm"  , "Q1"  , "Q2"  , "Q3"  , "Sum1"  , "Sum2"  , "Sum3",
                             "t_req_i", "tm_i", "Q1_i", "Q2_i", "Q3_i", "Sum1_i", "Sum2_i",
"Sum3_i"}
       out_rep_types = {"chart", "chart", "chart", "chart", "chart", "chart", "chart", "chart",
                                    "chart", "chart", "chart", "chart", "chart", "chart",
"chart", "chart"}
       c_set_shape_type(pointer, shape_type[1])
       c_set_in_types(pointer, in_types[1], in_types[2], in_types[3], in_types[4])
       c_set_in_names(pointer, in_names[1], in_names[2], in_names[3], in_names[4])
       c_set_in_defaults(pointer, in_defaults[1], in_defaults[2], in_defaults[3],
in_defaults[4])
       c_set_in_controls(pointer, in_controls[1], in_controls[2], in_controls[3],
in_controls[4])
       c_set_out_types(pointer, out_types[1], out_types[2],  out_types[3],  out_types[4],
out_types[5],  out_types[6],  out_types[7],  out_types[8],
                                                    out_types[9], out_types[10], out_types[11],
out_types[12], out_types[13], out_types[14], out_types[15], out_types[16])
       c_set_out_names(pointer, out_names[1], out_names[2],  out_names[3],  out_names[4],
out_names[5],  out_names[6],  out_names[7],  out_names[8],
                             out_names[9], out_names[10], out_names[11], out_names[12],
out_names[13], out_names[14], out_names[15], out_names[16])
       c_set_out_rep_types(pointer, out_rep_types[1],  out_rep_types[2],  out_rep_types[3],
out_rep_types[4],
```

```
                                        out_rep_types[5],  out_rep_types[6],  out_rep_types[7],
out_rep_types[8],
                                        out_rep_types[9],  out_rep_types[10], out_rep_types[11],
out_rep_types[12],
                                        out_rep_types[13], out_rep_types[14], out_rep_types[15],
out_rep_types[16])
        file:write("file_logic_A::head() end\n\n")
end
--***HEAD PABAIGA*****************************
--*******************************************
--*******************************************
--***PAGRINDINES FUNKCIJOS PRADZIA**************
function prepare(pointer_self, pointer_str)
        file:write("file_logic_A::prepare()\n")
        p_self = pointer_self
        p_v_str = pointer_str
        graph_size = c_get_graph_count(p_v_str)
        for i = 1, graph_size do
                graph_pointers[i] = {}
                graph_pointers[i]["pointer"] = c_get_graph_pointer(p_v_str, i - 1)
                graph_pointers[i]["p_str"] = c_get_pointer_as_str(graph_pointers[i]["pointer"], i
- 1)
                file:write(graph_pointers[i]["p_str"] .. "\n")
                graph_pointers[i]["shape"] = c_get_shape_type(graph_pointers[i]["pointer"], i - 1)
                file:write("pointer = " .. graph_pointers[i]["p_str"] .. "   shape = " ..
graph_pointers[i]["shape"] .."\n")
        end
        for i = 1, graph_size do
                ok = c_prepare(graph_pointers[i]["pointer"])
        end
        in_values[1], in_values[2], in_values[3], in_values[4], ok = c_get_init_vals(p_self, 4)
        pain     = graph_pointers[1]["pointer"]
        pump     = graph_pointers[2]["pointer"]
        integ1   = graph_pointers[3]["pointer"]
        integ2   = graph_pointers[4]["pointer"]
        integ3   = graph_pointers[5]["pointer"]
        sum1     = graph_pointers[6]["pointer"]
        sum2     = graph_pointers[7]["pointer"]
        sum3     = graph_pointers[8]["pointer"]
        pump_i   = graph_pointers[9]["pointer"]
        integ1_i = graph_pointers[10]["pointer"]
        integ2_i = graph_pointers[11]["pointer"]
        integ3_i = graph_pointers[12]["pointer"]
        sum1_i   = graph_pointers[13]["pointer"]
        sum2_i   = graph_pointers[14]["pointer"]
        sum3_i   = graph_pointers[15]["pointer"]
        Sim_time_h = in_values[1]
        dose_size  = in_values[2]
        dose_parts = in_values[3]
        compart_volume = in_values[4]
        file:write("file_logic_A::prepare() end\n\n")
end
function run_logic()
        file:write("file_logic_A::run_logic() \n")
        t_max = Sim_time_h * 60
        t = 0
        t_req = 0
        t_pred1 = math.huge
        t_pred2 = math.huge
        t_pred3 = math.huge
--SUSIGENERUOJAM T_REQ SKIRTUS POMPAI-----------------------------------------------
        t_req_arr = {}
        t_req_arr["size"] = 0
        file:write("generating t_req \n")
        while t < t_max do
                t_pain, ok = c_run_logic(pain, 1, "int", 0)
                t_pain = round(t_pain * 60, 4)
                t_pain = t_pain
                t = t + t_pain
                t_req_arr["size"] = t_req_arr["size"] + 1
                t_req_arr[t_req_arr["size"]] = t
```

```
                file:write("t_req = " .. t .. "\n")
        end
        file:write("done generating t_req \n")
-------------------------------------------------------------------------------
        t = 0
        req_count = 1
        t_req = t_req_arr[1]
        sigma1 = 0
        sigma2 = 0
        sigma3 = 0
        flux1 = 0
        flux2 = 0
        flux3 = 0
        Q_level1 = 0
        Q_level2 = 0
        Q_level3 = 0
        err1 = 0
        err2 = 0
        err3 = 0
        t_req_v = 0
        t_deliv_last = 0
        dose_size_ = dose_size / compart_volume
        while t < t_max and req_count < t_req_arr["size"] do
                t_req_v = 0
                min = find_min(t_req, t_pred1, t_pred2, t_pred3)
                file:write("t = " .. t .. ", t_req = " .. t_req .. ", t_pred1 = " .. t_pred1 .. ",
t_pred2 = " .. t_pred2 .. ", t_pred3 = " .. t_pred3 .. ", min = " .. min .. "\n")
                file:write("Q_level1 = " .. Q_level1 .. ", Q_level2 = " .. Q_level2 .. ", Q_level3
= " .. Q_level3 .. "\n")
                file:write("flux1 = " .. flux1 .. ", flux2 = " .. flux2 .. ", flux3 = " .. flux3
.. "\n\n")
                if         min == 1 then
                        t = t_req
                        t_deliv, p_state, ok = c_run_logic(pump, 2, "int", 0, "double", t)
                        if p_state == 0 then
                                t_req_v = 1
                                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4,
"double", t, "int", 1, "double", dose_size_)
                                if flux1 == 0 then
                                        t_pred1 = math.huge
                                        flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1,
4, "double", Q_level1, "double", Q_level2, "double", Q_level3)
                                        flux2, Q_level1, Q_level2, ok = c_run_logic(sum2, 3,
"double", Q_level1, "double", Q_level2)
                                        flux3, Q_level1, Q_level3, ok = c_run_logic(sum3, 3,
"double", Q_level1, "double", Q_level3)
                                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4,
"double", t, "int", 0, "double", flux1)
                                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4,
"double", t, "int", 0, "double", flux2)
                                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4,
"double", t, "int", 0, "double", flux3)
                                        sigma1 = round(sigma1, 4)
                                        sigma2 = round(sigma2, 4)
                                        sigma3 = round(sigma3, 4)
                                        if flux1 == 0 then
                                                t_pred1 = math.huge
                                        else
                                                t_pred1 = t + sigma1
                                        end
                                        if flux2 == 0 then
                                                t_pred2 = math.huge
                                        else
                                                t_pred2 = t + sigma2
                                        end
                                        if flux3 == 0 then
                                                t_pred3 = math.huge
                                        else
                                                t_pred3 = t + sigma3
                                        end
                                else
```

```
                                t_pred1 = t + sigma1
                        end
                        t_deliv_last = t_deliv
                end
                req_count = req_count + 1
                t_req = t_req_arr[req_count]
                t_req = round(t_req, 4)
        elseif min == 2 then
                t = t_pred1
                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4, "double", t,
"int", 2, "double", flux1)
                if t == t_pred2 then
                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4,
"double", t, "int", 2, "double", flux2)
                end
                if t == t_pred3 then
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4,
"double", t, "int", 2, "double", flux3)
                end
                flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1, 4, "double",
Q_level1, "double", Q_level2, "double", Q_level3)
                flux2, Q_level1, Q_level2, ok = c_run_logic(sum2, 3, "double", Q_level1,
"double", Q_level2)
                flux3, Q_level1, Q_level3, ok = c_run_logic(sum3, 3, "double", Q_level1,
"double", Q_level3)
                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4, "double", t,
"int", 0, "double", flux1)
                sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4, "double", t,
"int", 0, "double", flux2)
                sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4, "double", t,
"int", 0, "double", flux3)
                sigma1 = round(sigma1, 4)
                sigma2 = round(sigma2, 4)
                sigma3 = round(sigma3, 4)
                if flux1 == 0 then
                        t_pred1 = math.huge
                else
                        t_pred1 = t + sigma1
                end
                if flux2 == 0 then
                        t_pred2 = math.huge
                else
                        t_pred2 = t + sigma2
                end
                if flux3 == 0 then
                        t_pred3 = math.huge
                else
                        t_pred3 = t + sigma3
                end
        elseif min == 3 then
                t = t_pred2
                sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4, "double", t,
"int", 2, "double", flux2)
                if t == t_pred1 then
                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4,
"double", t, "int", 2, "double", flux1)
                end
                if t == t_pred3 then
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4,
"double", t, "int", 2, "double", flux3)
                end
                flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1, 4, "double",
Q_level1, "double", Q_level2, "double", Q_level3)
                flux2, Q_level1, Q_level2, ok = c_run_logic(sum2, 3, "double", Q_level1,
"double", Q_level2)
                flux3, Q_level1, Q_level3, ok = c_run_logic(sum3, 3, "double", Q_level1,
"double", Q_level3)
                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4, "double", t,
"int", 0, "double", flux1)
                sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4, "double", t,
"int", 0, "double", flux2)
```

```
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4, "double", t,
"int", 0, "double", flux3)
                        sigma1 = round(sigma1, 4)
                        sigma2 = round(sigma2, 4)
                        sigma3 = round(sigma3, 4)
                        if flux1 == 0 then
                                t_pred1 = math.huge
                        else
                                t_pred1 = t + sigma1
                        end
                        if flux2 == 0 then
                                t_pred2 = math.huge
                        else
                                t_pred2 = t + sigma2
                        end
                        if flux3 == 0 then
                                t_pred3 = math.huge
                        else
                                t_pred3 = t + sigma3
                        end
                elseif min == 4 then
                        t = t_pred3
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4, "double", t,
"int", 2, "double", flux3)
                        if t == t_pred2 then
                                sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4,
"double", t, "int", 2, "double", flux2)
                        end
                        if t == t_pred1 then
                                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4,
"double", t, "int", 2, "double", flux1)
                        end
                        flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1, 4, "double",
Q_level1, "double", Q_level2, "double", Q_level3)
                        flux2, Q_level1, Q_level2, ok = c_run_logic(sum2, 3, "double", Q_level1,
"double", Q_level2)
                        flux3, Q_level1, Q_level3, ok = c_run_logic(sum3, 3, "double", Q_level1,
"double", Q_level3)
                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4, "double", t,
"int", 0, "double", flux1)
                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4, "double", t,
"int", 0, "double", flux2)
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4, "double", t,
"int", 0, "double", flux3)
                        sigma1 = round(sigma1, 4)
                        sigma2 = round(sigma2, 4)
                        sigma3 = round(sigma3, 4)
                        if flux1 == 0 then
                                t_pred1 = math.huge
                        else
                                t_pred1 = t + sigma1
                        end
                        if flux2 == 0 then
                                t_pred2 = math.huge
                        else
                                t_pred2 = t + sigma2
                        end
                        if flux3 == 0 then
                                t_pred3 = math.huge
                        else
                                t_pred3 = t + sigma3
                        end
                end
                c_update_res(p_self, out_types[1], t_req_v, out_names[1])
                c_update_res(p_self, out_types[2], t, out_names[2])
                c_update_res(p_self, out_types[3], Q_level1, out_names[3])
                c_update_res(p_self, out_types[4], Q_level2, out_names[4])
                c_update_res(p_self, out_types[5], Q_level3, out_names[5])
                c_update_res(p_self, out_types[6], flux1, out_names[6])
                c_update_res(p_self, out_types[7], flux2, out_names[7])
                c_update_res(p_self, out_types[8], flux3, out_names[8])
```

```
	end
	file:write("*************************************************************************\n")
	file:write("*****BAIGEM PIRMA IMITACIJOS
DALI*************************************\n\n")
	c_run_logic(pump, 0, "int", 1, "double", 0)
	c_run_logic(pump, 0, "int", 2, "double", 0)
	sigma1 = 0
	sigma2 = 0
	sigma3 = 0
	flux1 = 0
	flux2 = 0
	flux3 = 0
	Q_level1 = 0
	Q_level2 = 0
	Q_level3 = 0
	err1 = 0
	err2 = 0
	err3 = 0
	t = 0
	t_req = 0
	t_pred1 = math.huge
	t_pred2 = math.huge
	t_pred3 = math.huge
	req_count = 1
	t_req = t_req_arr[1]
	t_req_v = 0 --
	t_deliv = 0 --
	t_deliv_last = 0
	t_req_pred = t_req_arr[1] --
	t_req_dose = math.huge    --
	pump_event = 0            --
	delta_t_deliv = 0                --
	dose_size_ = dose_size / (compart_volume * dose_parts)
	dose_left = 0  --
	t_last_dose_delvi = 0
	dose_size_to_deliv = dose_size_ --
	while t < t_max and req_count < t_req_arr["size"] do
		t_req_v = 0
		if t_req_pred <= t_req_dose then -- jei isorinis pompos ivykis ateina pirmiau nei
testinis pompos ivyks
			t_req = t_req_pred
			pump_event = 1
		else
			t_req = t_req_dose
			pump_event = 0
		end
		min = find_min(t_req, t_pred1, t_pred2, t_pred3)
		file:write("t = " .. t .. ", t_req = " .. t_req .. ", t_pred1 = " .. t_pred1 .. ",
t_pred2 = " .. t_pred2 .. ", t_pred3 = " .. t_pred3 .. ", min = " .. min .. "\n")
		file:write("Q_level1 = " .. Q_level1 .. ", Q_level2 = " .. Q_level2 .. ", Q_level3
= " .. Q_level3 .. "\n")
		file:write("flux1 = " .. flux1 .. ", flux2 = " .. flux2 .. ", flux3 = " .. flux3
.. "\n\n")
		if	 min == 1 then
			t = t_req
			if pump_event == 1 then
				t_req_v = -10
				t_deliv, p_state, ok = c_run_logic(pump_i, 2, "int", 0, "double",
t)
				delta_t_deliv = (t_deliv - t) / dose_parts
				if dose_left > 0 then
					dose_size_to_deliv = dose_size_ - dose_size_ * ((t_req_dose
- t_req_pred )/ delta_t_deliv)
				end
				dose_left = dose_parts - 1
				t_req_dose = t + delta_t_deliv
				req_count = req_count +1
				t_req_pred = t_req_arr[req_count]
				sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4,
"double", t, "int", 1, "double", dose_size_to_deliv)
				if flux1 == 0 then
```

```
                                        t_pred1 = math.huge
                                        flux1, Q_level1, Q_level2, Q_level3, ok =
c_run_logic(sum1_i, 4, "double", Q_level1, "double", Q_level2, "double", Q_level3)
                                        flux2, Q_level1, Q_level2, ok = c_run_logic(sum2_i, 3,
"double", Q_level1, "double", Q_level2)
                                        flux3, Q_level1, Q_level3, ok = c_run_logic(sum3_i, 3,
"double", Q_level1, "double", Q_level3)
                                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i,
4, "double", t, "int", 0, "double", flux1)
                                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i,
4, "double", t, "int", 0, "double", flux2)
                                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i,
4, "double", t, "int", 0, "double", flux3)
                                        sigma1 = round(sigma1, 4)
                                        sigma2 = round(sigma2, 4)
                                        sigma3 = round(sigma3, 4)
                                        if flux1 == 0 then
                                                t_pred1 = math.huge
                                        else
                                                t_pred1 = t + sigma1
                                        end
                                        if flux2 == 0 then
                                                t_pred2 = math.huge
                                        else
                                                t_pred2 = t + sigma2
                                        end
                                        if flux3 == 0 then
                                                t_pred3 = math.huge
                                        else
                                                t_pred3 = t + sigma3
                                        end
                                else
                                        t_pred1 = t + sigma1
                                end
                        else
                                t_req_v = -20
                                dose_size_to_deliv = dose_size_
                                dose_left = dose_left - 1
                                if dose_left == 0 then
                                        t_req_dose = math.huge
                                else
                                        t_req_dose = t + delta_t_deliv
                                end
                                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4,
"double", t, "int", 1, "double", dose_size_to_deliv)
                                if flux1 == 0 then
                                        t_pred1 = math.huge
                                        flux1, Q_level1, Q_level2, Q_level3, ok =
c_run_logic(sum1_i, 4, "double", Q_level1, "double", Q_level2, "double", Q_level3)
                                        flux2, Q_level1, Q_level2, ok = c_run_logic(sum2_i, 3,
"double", Q_level1, "double", Q_level2)
                                        flux3, Q_level1, Q_level3, ok = c_run_logic(sum3_i, 3,
"double", Q_level1, "double", Q_level3)
                                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i,
4, "double", t, "int", 0, "double", flux1)
                                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i,
4, "double", t, "int", 0, "double", flux2)
                                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i,
4, "double", t, "int", 0, "double", flux3)
                                        sigma1 = round(sigma1, 4)
                                        sigma2 = round(sigma2, 4)
                                        sigma3 = round(sigma3, 4)
                                        if flux1 == 0 then
                                                t_pred1 = math.huge
                                        else
                                                t_pred1 = t + sigma1
                                        end
                                        if flux2 == 0 then
                                                t_pred2 = math.huge
                                        else
                                                t_pred2 = t + sigma2
```

```
                                end
                                if flux3 == 0 then
                                        t_pred3 = math.huge
                                else
                                        t_pred3 = t + sigma3
                                end
                        else
                                t_pred1 = t + sigma1
                        end
                end
        elseif min == 2 then
                t = t_pred1
                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4, "double", t,
"int", 2, "double", flux1)
                if t == t_pred2 then
                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i, 4,
"double", t, "int", 2, "double", flux2)
                end
                if t == t_pred3 then
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i, 4,
"double", t, "int", 2, "double", flux3)
                end
                flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1_i, 4, "double",
Q_level1, "double", Q_level2, "double", Q_level3)
                flux2, Q_level1, Q_level2, ok = c_run_logic(sum2_i, 3, "double", Q_level1,
"double", Q_level2)
                flux3, Q_level1, Q_level3, ok = c_run_logic(sum3_i, 3, "double", Q_level1,
"double", Q_level3)
                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4, "double", t,
"int", 0, "double", flux1)
                sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i, 4, "double", t,
"int", 0, "double", flux2)
                sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i, 4, "double", t,
"int", 0, "double", flux3)
                sigma1 = round(sigma1, 4)
                sigma2 = round(sigma2, 4)
                sigma3 = round(sigma3, 4)
                if flux1 == 0 then
                        t_pred1 = math.huge
                else
                        t_pred1 = t + sigma1
                end
                if flux2 == 0 then
                        t_pred2 = math.huge
                else
                        t_pred2 = t + sigma2
                end
                if flux3 == 0 then
                        t_pred3 = math.huge
                else
                        t_pred3 = t + sigma3
                end
        elseif min == 3 then
                t = t_pred2
                sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i, 4, "double", t,
"int", 2, "double", flux2)
                if t == t_pred1 then
                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4,
"double", t, "int", 2, "double", flux1)
                end
                if t == t_pred3 then
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i, 4,
"double", t, "int", 2, "double", flux3)
                end
                flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1_i, 4, "double",
Q_level1, "double", Q_level2, "double", Q_level3)
                flux2, Q_level1, Q_level2, ok = c_run_logic(sum2_i, 3, "double", Q_level1,
"double", Q_level2)
                flux3, Q_level1, Q_level3, ok = c_run_logic(sum3_i, 3, "double", Q_level1,
"double", Q_level3)
```

```
                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4, "double", t,
"int", 0, "double", flux1)
                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i, 4, "double", t,
"int", 0, "double", flux2)
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i, 4, "double", t,
"int", 0, "double", flux3)
                        sigma1 = round(sigma1, 4)
                        sigma2 = round(sigma2, 4)
                        sigma3 = round(sigma3, 4)
                        if flux1 == 0 then
                                t_pred1 = math.huge
                        else
                                t_pred1 = t + sigma1
                        end
                        if flux2 == 0 then
                                t_pred2 = math.huge
                        else
                                t_pred2 = t + sigma2
                        end
                        if flux3 == 0 then
                                t_pred3 = math.huge
                        else
                                t_pred3 = t + sigma3
                        end
                elseif min == 4 then
                        t = t_pred3
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i, 4, "double", t,
"int", 2, "double", flux3)
                        if t == t_pred2 then
                                sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i, 4,
"double", t, "int", 2, "double", flux2)
                        end
                        if t == t_pred1 then
                                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4,
"double", t, "int", 2, "double", flux1)
                        end
                        flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1_i, 4, "double",
Q_level1, "double", Q_level2, "double", Q_level3)
                        flux2, Q_level1, Q_level2, ok = c_run_logic(sum2_i, 3, "double", Q_level1,
"double", Q_level2)
                        flux3, Q_level1, Q_level3, ok = c_run_logic(sum3_i, 3, "double", Q_level1,
"double", Q_level3)
                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4, "double", t,
"int", 0, "double", flux1)
                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i, 4, "double", t,
"int", 0, "double", flux2)
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i, 4, "double", t,
"int", 0, "double", flux3)
                        sigma1 = round(sigma1, 4)
                        sigma2 = round(sigma2, 4)
                        sigma3 = round(sigma3, 4)
                        if flux1 == 0 then
                                t_pred1 = math.huge
                        else
                                t_pred1 = t + sigma1
                        end
                        if flux2 == 0 then
                                t_pred2 = math.huge
                        else
                                t_pred2 = t + sigma2
                        end
                        if flux3 == 0 then
                                t_pred3 = math.huge
                        else
                                t_pred3 = t + sigma3
                        end
                end
                c_update_res(p_self, out_types[9] , t_req_v, out_names[9])
                c_update_res(p_self, out_types[10], t, out_names[10])
                c_update_res(p_self, out_types[11], Q_level1, out_names[11])
                c_update_res(p_self, out_types[12], Q_level2, out_names[12])
```

```
                c_update_res(p_self, out_types[13], Q_level3, out_names[13])
                c_update_res(p_self, out_types[14], flux1, out_names[14])
                c_update_res(p_self, out_types[15], flux2, out_names[15])
                c_update_res(p_self, out_types[16], flux3, out_names[16])
        end
        file:write("file_logic_A::run_logic() end\n")
        file:write("SIAM KARTUI TIEK\n\n")
end
function find_min(a, b, c, d)
        arr = {}
        arr[1] = a
        arr[2] = b
        arr[3] = c
        arr[4] = d
        min = arr[1]
        num = 1
        for i = 2, 4 do
                if min > arr[i] then
                        min = arr[i]
                        num = i
                end
        end
        return num
end
function round(number, digits)
    return tonumber(string.format("%." .. (digits or 0) .. "f", number))
end
function end_work()
        file:write("lua baige darba\n")
        file:close()
end
function kill(ok, err_msg, method, message)
        if ok == 0 then
                if err_msg == "" then
                else
                        file:write(err_msg .. "\n")
                end
                if methos == "" then
                else
                        file:write(method .." end\n")
                end
                if message == "" then
                else
                        file:write(message .. "\n")
                end
                return true
        end
        return false
end
--***PAGRINDINES FUNKCIJOS PABAIGA**************
--*****************************************
```

7.10. Failas lua_logic_B.lua

```
--[[*************************************************************************
        ****ZEMIAU ESANCIO TEKSTO NEKEISTI*************************************]]--
function exists(n)
                local f = io.open(n)
                if f == nil then
                        return false
                else
                        f = io.close()
                        return true
                end
end
f_num = 0
t = true
f_name = os.time() .. "_lua_logic_log_" .. f_num .. ".txt"
while t do
        if exists(f_name) then
```

```
                f_num = f_num + 1
                f_name = os.time() .. "_lua_logic_log_" .. f_num .. ".txt"
        else
                t = false
        end
end
local file = io.open(f_name, "w")
file:write("lua pradejo darba\n")
--[[*********************************************************************
        ****ZEMIAU ESANTI TEKSTA GALIMA KEISTI**********************************]]--
local p_self
local p_v_str
local in_names
local in_values = {}
local out_names
local out_values = {}
local graph_size
local graph_pointers = {}
local adj_list = {}
local shift_list = {}
local pain                      -- 1
local pump                      -- 2
local pump_i            -- 9
local integ1            -- 3
local integ2            -- 4
local integ3            -- 5
local integ1_i                  --10
local integ2_i          --11
local integ3_i          --12
local sum1                      -- 6
local sum2                      -- 7
local sum3                      -- 8
local sum1_i            --13
local sum2_i            --14
local sum3_i            --15
local Sim_time_h
local dose_size
local dose_parts
local compart_volume
--*******************************************
--***HEAD PRADZIA*****************************
function head(pointer)                                          --pointer - rodykle i c_object tipo
c++ objekta
        file:write("file_logic::head()\n")
        shape_type = {"logic"}
        in_types = {"double", "double", "double", "double"}
        in_names = {"Sim_time_h", "Dose_size", "Dose_parts", "Compart_volume"}
        in_defaults = {12, 1000, 10, 13}
        in_controls = {"text_box", "text_box", "text_box", "text_box"}
        out_types = {"double", "double", "double", "double", "double", "double", "double",
"double",
                                "double", "double", "double", "double", "double", "double",
"double", "double"}
        out_names = {"t_req"  , "tm"  , "Q1"  , "Q2"  , "Q3"  , "Sum1"  , "Sum2"  , "Sum3",
                                "t_req_i", "tm_i", "Q1_i", "Q2_i", "Q3_i", "Sum1_i", "Sum2_i",
"Sum3_i"}
        out_rep_types = {"chart", "chart", "chart", "chart", "chart", "chart", "chart", "chart",
                                        "chart", "chart", "chart", "chart", "chart", "chart",
"chart", "chart"}
        c_set_shape_type(pointer, shape_type[1])
        c_set_in_types(pointer, in_types[1], in_types[2], in_types[3], in_types[4])
        c_set_in_names(pointer, in_names[1], in_names[2], in_names[3], in_names[4])
        c_set_in_defaults(pointer, in_defaults[1], in_defaults[2], in_defaults[3],
in_defaults[4])
        c_set_in_controls(pointer, in_controls[1], in_controls[2], in_controls[3],
in_controls[4])
        c_set_out_types(pointer, out_types[1], out_types[2],  out_types[3],  out_types[4],
out_types[5],  out_types[6],  out_types[7],  out_types[8],
                                                        out_types[9], out_types[10], out_types[11],
out_types[12], out_types[13], out_types[14], out_types[15], out_types[16])
```

```
	c_set_out_names(pointer, out_names[1], out_names[2],  out_names[3],  out_names[4],
out_names[5],  out_names[6],  out_names[7],  out_names[8],
                                out_names[9], out_names[10], out_names[11], out_names[12],
out_names[13], out_names[14], out_names[15], out_names[16])
	c_set_out_rep_types(pointer, out_rep_types[1],  out_rep_types[2],  out_rep_types[3],
out_rep_types[4],
                                        out_rep_types[5],  out_rep_types[6],  out_rep_types[7],
out_rep_types[8],
                                        out_rep_types[9],  out_rep_types[10], out_rep_types[11],
out_rep_types[12],
                                        out_rep_types[13], out_rep_types[14], out_rep_types[15],
out_rep_types[16])
	file:write("file_logic_B::head() end\n\n")
end
--***HEAD PABAIGA*****************************
--*******************************************
--*******************************************
--***PAGRINDINES FUNKCIJOS PRADZIA**************
function prepare(pointer_self, pointer_str)
	file:write("file_logic_B::prepare()\n")
	--sudarom reikiamas strukturas, inicializuojam kintamuosius
	p_self = pointer_self
	p_v_str = pointer_str
	graph_size = c_get_graph_count(p_v_str)
	for i = 1, graph_size do
		graph_pointers[i] = {}
		graph_pointers[i]["pointer"] = c_get_graph_pointer(p_v_str, i - 1)
		graph_pointers[i]["p_str"] = c_get_pointer_as_str(graph_pointers[i]["pointer"], i
- 1)
		file:write(graph_pointers[i]["p_str"] .. "\n")
		graph_pointers[i]["shape"] = c_get_shape_type(graph_pointers[i]["pointer"], i - 1)
		file:write("pointer = " .. graph_pointers[i]["p_str"] .. "   shape = " ..
graph_pointers[i]["shape"] .."\n")
	end
	for i = 1, graph_size do
		ok = c_prepare(graph_pointers[i]["pointer"])
	end
	in_values[1], in_values[2], in_values[3], in_values[4], ok = c_get_init_vals(p_self, 4)
	pain     = graph_pointers[1]["pointer"]
	pump     = graph_pointers[2]["pointer"]
	integ1   = graph_pointers[3]["pointer"]
	integ2   = graph_pointers[4]["pointer"]
	integ3   = graph_pointers[5]["pointer"]
	sum1     = graph_pointers[6]["pointer"]
	sum2     = graph_pointers[7]["pointer"]
	sum3     = graph_pointers[8]["pointer"]
	pump_i   = graph_pointers[9]["pointer"]
	integ1_i = graph_pointers[10]["pointer"]
	integ2_i = graph_pointers[11]["pointer"]
	integ3_i = graph_pointers[12]["pointer"]
	sum1_i   = graph_pointers[13]["pointer"]
	sum2_i   = graph_pointers[14]["pointer"]
	sum3_i   = graph_pointers[15]["pointer"]
	Sim_time_h = in_values[1]
	dose_size  = in_values[2]
	dose_parts = in_values[3]
	compart_volume = in_values[4]
	file:write("file_logic_B::prepare() end\n\n")
end
function run_logic()
	file:write("file_logic_B::run_logic() \n")
	t_max = Sim_time_h * 60
	t = 0
	t_req = 0
	t_pred1 = math.huge
	t_pred2 = math.huge
	t_pred3 = math.huge
	t = 0
	req_count = 1
	t_req = 0
	sigma1 = 0
```

```
	sigma2 = 0
	sigma3 = 0
	flux1 = 0
	flux2 = 0
	flux3 = 0
	Q_level1 = 0
	Q_level2 = 0
	Q_level3 = 0
	err1 = 0
	err2 = 0
	err3 = 0
	t_req_v = 0
	t_deliv_last = 0
	dose_size_ = dose_size / compart_volume
	t_pain, ok = c_run_logic(pain, 1, "double", Q_level2)
	t_req = t_pain
	while t < t_max do
		t_req_v = 0
		min = find_min(t_req, t_pred1, t_pred2, t_pred3)
		file:write("t = " .. t .. ", t_req = " .. t_req .. ", t_pred1 = " .. t_pred1 .. ",
t_pred2 = " .. t_pred2 .. ", t_pred3 = " .. t_pred3 .. ", min = " .. min .. "\n")
		file:write("Q_level1 = " .. Q_level1 .. ", Q_level2 = " .. Q_level2 .. ", Q_level3
= " .. Q_level3 .. "\n")
		file:write("flux1 = " .. flux1 .. ", flux2 = " .. flux2 .. ", flux3 = " .. flux3
.. "\n\n")
		if	min == 1 then
			t = t_req
			t_deliv, p_state, ok = c_run_logic(pump, 2, "int", 0, "double", t)
			if p_state == 0 then
				t_req_v = -20
				sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4,
"double", t, "int", 1, "double", dose_size_)
				if flux1 == 0 then
					t_pred1 = math.huge
					flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1,
4, "double", Q_level1, "double", Q_level2, "double", Q_level3)
					flux2, Q_level1, Q_level2, ok = c_run_logic(sum2, 3,
"double", Q_level1, "double", Q_level2)
					flux3, Q_level1, Q_level3, ok = c_run_logic(sum3, 3,
"double", Q_level1, "double", Q_level3)
					sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4,
"double", t, "int", 0, "double", flux1)
					sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4,
"double", t, "int", 0, "double", flux2)
					sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4,
"double", t, "int", 0, "double", flux3)
					sigma1 = round(sigma1, 4)
					sigma2 = round(sigma2, 4)
					sigma3 = round(sigma3, 4)
					if flux1 == 0 then
						t_pred1 = math.huge
					else
						t_pred1 = t + sigma1
					end
					if flux2 == 0 then
						t_pred2 = math.huge
					else
						t_pred2 = t + sigma2
					end
					if flux3 == 0 then
						t_pred3 = math.huge
					else
						t_pred3 = t + sigma3
					end
				else
					t_pred1 = t + sigma1
				end
				t_deliv_last = t_deliv
			end
			req_count = req_count + 1
			t_req = math.huge
```

```
		elseif min == 2 then
			t = t_pred1
			sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4, "double", t,
"int", 2, "double", flux1)
			if t == t_pred2 then
				sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4,
"double", t, "int", 2, "double", flux2)
			end
			if t == t_pred3 then
				sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4,
"double", t, "int", 2, "double", flux3)
			end
			flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1, 4, "double",
Q_level1, "double", Q_level2, "double", Q_level3)
			flux2, Q_level1, Q_level2, ok = c_run_logic(sum2, 3, "double", Q_level1,
"double", Q_level2)
			flux3, Q_level1, Q_level3, ok = c_run_logic(sum3, 3, "double", Q_level1,
"double", Q_level3)
			sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4, "double", t,
"int", 0, "double", flux1)
			sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4, "double", t,
"int", 0, "double", flux2)
			sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4, "double", t,
"int", 0, "double", flux3)
			sigma1 = round(sigma1, 4)
			sigma2 = round(sigma2, 4)
			sigma3 = round(sigma3, 4)
			if flux1 == 0 then
				t_pred1 = math.huge
			else
				t_pred1 = t + sigma1
			end
			if flux2 == 0 then
				t_pred2 = math.huge
			else
				t_pred2 = t + sigma2
			end
			if flux3 == 0 then
				t_pred3 = math.huge
			else
				t_pred3 = t + sigma3
			end
		elseif min == 3 then
			t = t_pred2
			sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4, "double", t,
"int", 2, "double", flux2)
			if t == t_pred1 then
				sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4,
"double", t, "int", 2, "double", flux1)
			end
			if t == t_pred3 then
				sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4,
"double", t, "int", 2, "double", flux3)
			end
			t_pain, ok = c_run_logic(pain, 1, "double", Q_level2)
			if t_pain > -1 then
				if t_req == math.huge then
					t_req = t + t_pain
				end
			end
			flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1, 4, "double",
Q_level1, "double", Q_level2, "double", Q_level3)
			flux2, Q_level1, Q_level2, ok = c_run_logic(sum2, 3, "double", Q_level1,
"double", Q_level2)
			flux3, Q_level1, Q_level3, ok = c_run_logic(sum3, 3, "double", Q_level1,
"double", Q_level3)
			sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4, "double", t,
"int", 0, "double", flux1)
			sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4, "double", t,
"int", 0, "double", flux2)
```

```
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4, "double", t,
"int", 0, "double", flux3)
                        sigma1 = round(sigma1, 4)
                        sigma2 = round(sigma2, 4)
                        sigma3 = round(sigma3, 4)
                        if flux1 == 0 then
                                t_pred1 = math.huge
                        else
                                t_pred1 = t + sigma1
                        end
                        if flux2 == 0 then
                                t_pred2 = math.huge
                        else
                                t_pred2 = t + sigma2
                        end
                        if flux3 == 0 then
                                t_pred3 = math.huge
                        else
                                t_pred3 = t + sigma3
                        end
                elseif min == 4 then
                        t = t_pred3
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4, "double", t,
"int", 2, "double", flux3)
                        if t == t_pred2 then
                                sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4,
"double", t, "int", 2, "double", flux2)
                        end
                        if t == t_pred1 then
                                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4,
"double", t, "int", 2, "double", flux1)
                        end
                        flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1, 4, "double",
Q_level1, "double", Q_level2, "double", Q_level3)
                        flux2, Q_level1, Q_level2, ok = c_run_logic(sum2, 3, "double", Q_level1,
"double", Q_level2)
                        flux3, Q_level1, Q_level3, ok = c_run_logic(sum3, 3, "double", Q_level1,
"double", Q_level3)
                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1, 4, "double", t,
"int", 0, "double", flux1)
                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2, 4, "double", t,
"int", 0, "double", flux2)
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3, 4, "double", t,
"int", 0, "double", flux3)
                        sigma1 = round(sigma1, 4)
                        sigma2 = round(sigma2, 4)
                        sigma3 = round(sigma3, 4)
                        if flux1 == 0 then
                                t_pred1 = math.huge
                        else
                                t_pred1 = t + sigma1
                        end
                        if flux2 == 0 then
                                t_pred2 = math.huge
                        else
                                t_pred2 = t + sigma2
                        end
                        if flux3 == 0 then
                                t_pred3 = math.huge
                        else
                                t_pred3 = t + sigma3
                        end
                end
                c_update_res(p_self, out_types[1], t_req_v, out_names[1])
                c_update_res(p_self, out_types[2], t, out_names[2])
                c_update_res(p_self, out_types[3], Q_level1, out_names[3])
                c_update_res(p_self, out_types[4], Q_level2, out_names[4])
                c_update_res(p_self, out_types[5], Q_level3, out_names[5])
                c_update_res(p_self, out_types[6], flux1, out_names[6])
                c_update_res(p_self, out_types[7], flux2, out_names[7])
                c_update_res(p_self, out_types[8], flux3, out_names[8])
```

```
	end
	file:write("*************************************************************************\n")
	file:write("*****BAIGEM PIRMA IMITACIJOS
DALI*************************************\n\n")
	c_run_logic(pump, 0, "int", 1, "double", 0)
	c_run_logic(pump, 0, "int", 2, "double", 0)
	sigma1 = 0
	sigma2 = 0
	sigma3 = 0
	flux1 = 0
	flux2 = 0
	flux3 = 0
	Q_level1 = 0
	Q_level2 = 0
	Q_level3 = 0
	err1 = 0
	err2 = 0
	err3 = 0
	t = 0
	t_req = 0
	t_pred1 = math.huge
	t_pred2 = math.huge
	t_pred3 = math.huge
	t_pain, ok = c_run_logic(pain, 1, "double", Q_level2)
	req_count = 1
	t_req = t_pain
	t_req_v = 0 --
	t_deliv = 0 --
	t_deliv_last = 0
	t_req_pred = t_pain --
	t_req_dose = math.huge    --
	pump_event = 0            --
	delta_t_deliv = 0               --
	dose_size_ = dose_size / (compart_volume * dose_parts)
	dose_left = 0  --
	t_last_dose_delvi = 0
	dose_size_to_deliv = dose_size_ --
	while t < t_max do
		t_req_v = 0
		if t_req_pred <= t_req_dose then -- jei isorinis pompos ivykis ateina pirmiau nei
testinis pompos ivyks
			t_req = t_req_pred
			pump_event = 1
		else
			t_req = t_req_dose
			pump_event = 0
		end
		min = find_min(t_req, t_pred1, t_pred2, t_pred3)
		file:write("t = " .. t .. ", t_req = " .. t_req .. ", t_pred1 = " .. t_pred1 .. ",
t_pred2 = " .. t_pred2 .. ", t_pred3 = " .. t_pred3 .. ", min = " .. min .. "\n")
		file:write("Q_level1 = " .. Q_level1 .. ", Q_level2 = " .. Q_level2 .. ", Q_level3
= " .. Q_level3 .. "\n")
		file:write("flux1 = " .. flux1 .. ", flux2 = " .. flux2 .. ", flux3 = " .. flux3
.. "\n\n")
		if         min == 1 then
			t = t_req
			if pump_event == 1 then
				t_req_v = -10
				t_deliv, p_state, ok = c_run_logic(pump_i, 2, "int", 0, "double",
t)
				delta_t_deliv = (t_deliv - t) / dose_parts
				if dose_left > 0 then
					dose_size_to_deliv = dose_size_ - dose_size_ * ((t_req_dose
- t_req_pred )/ delta_t_deliv)
				end
				dose_left = dose_parts - 1
				t_req_dose = t + delta_t_deliv
				req_count = req_count +1
				t_req_pred = math.huge
				sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4,
"double", t, "int", 1, "double", dose_size_to_deliv)
```

```
                                if flux1 == 0 then
                                        t_pred1 = math.huge
                                        flux1, Q_level1, Q_level2, Q_level3, ok =
c_run_logic(sum1_i, 4, "double", Q_level1, "double", Q_level2, "double", Q_level3)
                                        flux2, Q_level1, Q_level2, ok = c_run_logic(sum2_i, 3,
"double", Q_level1, "double", Q_level2)
                                        flux3, Q_level1, Q_level3, ok = c_run_logic(sum3_i, 3,
"double", Q_level1, "double", Q_level3)
                                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i,
4, "double", t, "int", 0, "double", flux1)
                                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i,
4, "double", t, "int", 0, "double", flux2)
                                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i,
4, "double", t, "int", 0, "double", flux3)
                                        sigma1 = round(sigma1, 4)
                                        sigma2 = round(sigma2, 4)
                                        sigma3 = round(sigma3, 4)
                                        if flux1 == 0 then
                                                t_pred1 = math.huge
                                        else
                                                t_pred1 = t + sigma1
                                        end
                                        if flux2 == 0 then
                                                t_pred2 = math.huge
                                        else
                                                t_pred2 = t + sigma2
                                        end
                                        if flux3 == 0 then
                                                t_pred3 = math.huge
                                        else
                                                t_pred3 = t + sigma3
                                        end
                                else
                                        t_pred1 = t + sigma1
                                end
                        else
                                t_req_v = -20
                                dose_size_to_deliv = dose_size_
                                dose_left = dose_left - 1
                                if dose_left == 0 then
                                        t_req_dose = math.huge
                                else
                                        t_req_dose = t + delta_t_deliv
                                end
                                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4,
"double", t, "int", 1, "double", dose_size_to_deliv)
                                if flux1 == 0 then
                                        t_pred1 = math.huge
                                        flux1, Q_level1, Q_level2, Q_level3, ok =
c_run_logic(sum1_i, 4, "double", Q_level1, "double", Q_level2, "double", Q_level3)
                                        flux2, Q_level1, Q_level2, ok = c_run_logic(sum2_i, 3,
"double", Q_level1, "double", Q_level2)
                                        flux3, Q_level1, Q_level3, ok = c_run_logic(sum3_i, 3,
"double", Q_level1, "double", Q_level3)
                                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i,
4, "double", t, "int", 0, "double", flux1)
                                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i,
4, "double", t, "int", 0, "double", flux2)
                                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i,
4, "double", t, "int", 0, "double", flux3)
                                        sigma1 = round(sigma1, 4)
                                        sigma2 = round(sigma2, 4)
                                        sigma3 = round(sigma3, 4)
                                        if flux1 == 0 then
                                                t_pred1 = math.huge
                                        else
                                                t_pred1 = t + sigma1
                                        end
                                        if flux2 == 0 then
                                                t_pred2 = math.huge
                                        else
```

```
                                        t_pred2 = t + sigma2
                                end
                                if flux3 == 0 then
                                        t_pred3 = math.huge
                                else
                                        t_pred3 = t + sigma3
                                end
                        else
                                t_pred1 = t + sigma1
                        end
                end
        elseif min == 2 then
                t = t_pred1
                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4, "double", t,
"int", 2, "double", flux1)
                if t == t_pred2 then
                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i, 4,
"double", t, "int", 2, "double", flux2)
                end
                if t == t_pred3 then
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i, 4,
"double", t, "int", 2, "double", flux3)
                end
                flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1_i, 4, "double",
Q_level1, "double", Q_level2, "double", Q_level3)
                flux2, Q_level1, Q_level2, ok = c_run_logic(sum2_i, 3, "double", Q_level1,
"double", Q_level2)
                flux3, Q_level1, Q_level3, ok = c_run_logic(sum3_i, 3, "double", Q_level1,
"double", Q_level3)
                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4, "double", t,
"int", 0, "double", flux1)
                sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i, 4, "double", t,
"int", 0, "double", flux2)
                sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i, 4, "double", t,
"int", 0, "double", flux3)
                sigma1 = round(sigma1, 4)
                sigma2 = round(sigma2, 4)
                sigma3 = round(sigma3, 4)
                if flux1 == 0 then
                        t_pred1 = math.huge
                else
                        t_pred1 = t + sigma1
                end
                if flux2 == 0 then
                        t_pred2 = math.huge
                else
                        t_pred2 = t + sigma2
                end
                if flux3 == 0 then
                        t_pred3 = math.huge
                else
                        t_pred3 = t + sigma3
                end
        elseif min == 3 then
                t = t_pred2
                sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i, 4, "double", t,
"int", 2, "double", flux2)
                if t == t_pred1 then
                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4,
"double", t, "int", 2, "double", flux1)
                end
                if t == t_pred3 then
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i, 4,
"double", t, "int", 2, "double", flux3)
                end
                t_pain, ok = c_run_logic(pain, 1, "double", Q_level2)
                if t_pain > -1 then
                        if t_req_pred == math.huge then
                                t_req_pred = t + t_pain
                        end
                end
```

```
                        flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1_i, 4, "double",
Q_level1, "double", Q_level2, "double", Q_level3)
                        flux2, Q_level1, Q_level2, ok = c_run_logic(sum2_i, 3, "double", Q_level1,
"double", Q_level2)
                        flux3, Q_level1, Q_level3, ok = c_run_logic(sum3_i, 3, "double", Q_level1,
"double", Q_level3)
                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4, "double", t,
"int", 0, "double", flux1)
                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i, 4, "double", t,
"int", 0, "double", flux2)
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i, 4, "double", t,
"int", 0, "double", flux3)
                        sigma1 = round(sigma1, 4)
                        sigma2 = round(sigma2, 4)
                        sigma3 = round(sigma3, 4)
                        if flux1 == 0 then
                                t_pred1 = math.huge
                        else
                                t_pred1 = t + sigma1
                        end
                        if flux2 == 0 then
                                t_pred2 = math.huge
                        else
                                t_pred2 = t + sigma2
                        end
                        if flux3 == 0 then
                                t_pred3 = math.huge
                        else
                                t_pred3 = t + sigma3
                        end
                elseif min == 4 then
                        t = t_pred3
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i, 4, "double", t,
"int", 2, "double", flux3)
                        if t == t_pred2 then
                                sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i, 4,
"double", t, "int", 2, "double", flux2)
                        end
                        if t == t_pred1 then
                                sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4,
"double", t, "int", 2, "double", flux1)
                        end
                        flux1, Q_level1, Q_level2, Q_level3, ok = c_run_logic(sum1_i, 4, "double",
Q_level1, "double", Q_level2, "double", Q_level3)
                        flux2, Q_level1, Q_level2, ok = c_run_logic(sum2_i, 3, "double", Q_level1,
"double", Q_level2)
                        flux3, Q_level1, Q_level3, ok = c_run_logic(sum3_i, 3, "double", Q_level1,
"double", Q_level3)
                        sigma1, flux1, Q_level1, err1, ok = c_run_logic(integ1_i, 4, "double", t,
"int", 0, "double", flux1)
                        sigma2, flux2, Q_level2, err2, ok = c_run_logic(integ2_i, 4, "double", t,
"int", 0, "double", flux2)
                        sigma3, flux3, Q_level3, err3, ok = c_run_logic(integ3_i, 4, "double", t,
"int", 0, "double", flux3)
                        sigma1 = round(sigma1, 4)
                        sigma2 = round(sigma2, 4)
                        sigma3 = round(sigma3, 4)
                        if flux1 == 0 then
                                t_pred1 = math.huge
                        else
                                t_pred1 = t + sigma1
                        end
                        if flux2 == 0 then
                                t_pred2 = math.huge
                        else
                                t_pred2 = t + sigma2
                        end
                        if flux3 == 0 then
                                t_pred3 = math.huge
                        else
                                t_pred3 = t + sigma3
```

```
                        end
                end
                c_update_res(p_self, out_types[9] , t_req_v, out_names[9])
                c_update_res(p_self, out_types[10], t, out_names[10])
                c_update_res(p_self, out_types[11], Q_level1, out_names[11])
                c_update_res(p_self, out_types[12], Q_level2, out_names[12])
                c_update_res(p_self, out_types[13], Q_level3, out_names[13])
                c_update_res(p_self, out_types[14], flux1, out_names[14])
                c_update_res(p_self, out_types[15], flux2, out_names[15])
                c_update_res(p_self, out_types[16], flux3, out_names[16])
        end
        file:write("file_logic::run_logic() end\n")
        file:write("SIAM KARTUI TIEK\n\n")
end
function find_min(a, b, c, d)
        arr = {}
        arr[1] = a
        arr[2] = b
        arr[3] = c
        arr[4] = d
        min = arr[1]
        num = 1
        for i = 2, 4 do
                if min > arr[i] then
                        min = arr[i]
                        num = i
                end
        end
        return num
end
function round(number, digits)
    return tonumber(string.format("%." .. (digits or 0) .. "f", number))
end
function end_work()
        file:write("lua baige darba\n")
        file:close()
end
function kill(ok, err_msg, method, message)
        if ok == 0 then
                if err_msg == "" then
                else
                        file:write(err_msg .. "\n")
                end
                if methos == "" then
                else
                        file:write(method .." end\n")
                end
                if message == "" then
                else
                        file:write(message .. "\n")
                end
                return true
        end
        return false
end
--***PAGRINDINES FUNKCIJOS PABAIGA**************
--*****************************************
```