KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Donatas Urbas

Ląstelių tinklų imitacinis modeliavimas

Magistro baigiamasis darbas

Darbo vadovas
prof. hab. dr. Henrikas Pranevičius

KAUNAS, 2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Donatas Urbas

Ląstelių tinklų imitacinis modeliavimas

Magistro baigiamasis darbas

Recenzentas:
prof. Eduardas Bareiša
2011-05-30

Darbo vadovas:
prof. hab. dr. Henrikas Pranevičius
2011-05-30

Autorius: IFM – 9/2 gr. studentas
Donatas Urbas
2011-05-30

KAUNAS, 2011

Summary

This document contains an analysis of gap junction simulation model, based on aggregate simulation method. The aim is to find out does aggregate simulation method is appropriate for creation of cells network simulation models. Find out the electric signal of cells network where gap junction is placed as linear and non-linear element.

Quantized state method was used to make a research. Was created an aggregate cells network simulation model, which components formally are described using Piece Linear Aggregates (PLA). In order to implement the simulation PCA model, the program for aggregate systems modeling was used (authors: Virginijus Pampikas and Donatas Urbas).

The interaction between two cells was analyzed when the gap junction was implemented through linear and non-linear element. The analysis of the influence of the change of gap junction resistance for transfer of the electric signals was made.

Santrauka

Šiame darbe atliekamas ląstelių tinklo plyšinės jungties imitacinio modelio tyrimas taikant agregatinį modeliavimo metodą. Siekiama nustatyti ar agregatinis metodas tinkamas ląstelių tinklų imitaciniams modeliams kurti. Taip pat tiriama elektros srovė ląstelių tinkle, kai plyšinės jungties varža realizuota tiesiniu ir netiesiniu elementu.

Tyrimui atlikti naudojamas kvantuotų sistemos būsenų modelis. Buvo sudarytas agregatinis imitacinis ląstelių tinklo modelis, kurio komponentai aprašyti naudojant PLA formalizmą. Imitacinis modelis realizuotas panaudojant agregatinių sistemų modeliavimo programą (autoriai: Virginijus Pampikas ir Donatas Urbas). Modeliavimo programos dokumentas pateiktas magistrinio darbo tarpinėse ataskaitose.

Buvo tirta dviejų ląstelių sąveika, kai plyšinė jungtis realizuota tiek tiesiniu, tiek netiesiniu elementu. Ištirta plyšinės jungties varžos kitimo įtaka elektrinių signalų perdavimui.

Turinys

Iliustracijų sąrašas

Lentelių sąrašas

ĮVADAS

Kaskart dideliu greičiu tobulėjant technologijoms jos taikomos įvairiose srityse: versle, automatizuojamos įvairios mūsų gyvenimo sritys (bankai, bibliotekos, įvairios registracijos įstaigose), modeliuojami įvairūs realiai vykstantys procesai, taip pat platus taikymas ir medicinoje.

Technologijoms tobulėjant žmonės gali detaliai nagrinėti organizmų, atskirų ląstelių sandaras, ryšius tarp ląstelių ir juos aprašyti. Tokiais atvejais atsiranda poreikis modeliuoti įvairias tarpląstelines sąveikas, ryšius tarp ląstelių. Tokiose sistemose stebimi įvairiausi parametrai (tiek išoriškai veikiantys sistemą, tiek ir vidiniai), analizuojami sistemos dėsningumai, elgsena. Tokiose mikrosistemose svarbu ne tik stebėti realiai vykstančius procesus, bet ir juos sugebėti sumodeliuoti kompiuteriu. Taip galima pasiekti norimą analizuojamos sistemos detalizacijos lygį, analizuoti norimo dydžio sistemą.

Darbo tikslas yra sumodeliuoti plyšinių jungčių tinklą turint atskirus tinklo komponenčių modelius. Plyšinių jungčių sistema detalizuojama iki vienos jungties bei sąveikos, kai plyšinės jungtys jungiasi tarpusavyje. Turint tokias atskiras komponentes (modelius) galima sudarinėti norimas struktūras, tiek plokštumines, tiek erdvines.

Biomedicina labai svarbi mokslo šaka jungianti daugelį mokslo sričių: mediciną, matematiką, elektros grandinių teoriją, informatiką. Toks daugialypis ryšys verčia skirtingų mokslo krypčių žinias susieti. Taikant taikomosios matematikos, informatikos, statistikos bei elektros grandinių mokslo žinias aprašyti biologines problemas. Analizuojama problema bus aprašyta remiantis jau žinomais elektros grandinėse vykstančiais procesais. Šiems procesams apibrėžti sudaryta PLA formali specifikacija, o tiriamoji dalis realizuota specialiai tam sukurta agregatų imitacine modeliavimo sistema.

1. TIKSLAI IR UŽDAVINIAI

1.1. Dokumento paskirtis

Kompiuterinis biologinių procesų modeliavimas – neatsiejama šiuolaikinės medicinos kryptis. Nuolat atliekami detalūs tyrimai siekiant išsiaiškinti procesus vykstančius mažiausiuose biologiniuose objektuose (ląstelių lygmuo, jų skaidymas). Tačiau tokie tyrimai reikalauja didelių kaštų, o tokiais tyrimais nustatomi esminiai objektų veikimo principai. Žinant šiuos principus tikslinga sudaryti imitacinius kompiuterinius modelius. Reikalingas įrankis analizuoti esamai sričiai, kuriuo galėtų naudotis didesnis skaičius mokslininkų ar studentų.

1.2. Tikslas

Sukurti ląstelių tinkle sklindančio elektrinio signalo imitatorių ir jį panaudojant atlikti elektrinio signalo sklidimo imitaciją sąveikaujant dviem ląstelėms.

1.3. Uždaviniai

1) Patobulinti plyšinės jungties modelį, kai plyšinės jungties varža yra netiesinis elementas;
2) Aprašyti imitacinį neuronų tinklo struktūros modelį;
3) Taikant agregatinį sprendimo metodą ir panaudojant sukurtą agregatais paremtą imitavimo sistemą sumodeliuoti elektrinio signalo perdavimą tarp dviejų ląstelių, kurias sieja plyšinė jungtis.

2. RAKTINIAI ŽODŽIAI

Lentelė 1 Terminų žodynėlis

Eil. Nr.	Terminas lietuviškai	Terminas angliškai	Paaiškinimas
1.	Plyšinė jungtis	Gap junction	Jungtis tarp dviejų ląstelių membranų
2.	Plyšinės jungties kanalas	Gap junction channel	Plyšinės jungties sudedamoji dalis
3.	Metabolinis signalas		Jonų perdavimas
4.	Koneksinai	Connexons	Baltymai iš kurių sudarytas plyšinės jungties kanalas
5.	Imitacinis modelis	Simutation model	Realybę tam tikru tikslumu atvaizduojanti kompiuterinė sistema
6.	GO	Gate open	Atviri vartai
7.	GC	Gate close	Uždari vartai
8.	Homomeriniai puskanaliai		Puskanaliai, sudaryti iš vieno tipo koneksinų
9.	Heteromeriniai puskanaliai		Puskanaliai, sudaryti iš daugiau negu vieno tipo koneksinų
10.	Homotipiniai plyšinės jungties kanalai		Kanalai, sudaryti iš 2 vienodų puskanalių
11.	Heterotipiniai plyšinės jungties kanalai		Kanalai, sudaryti iš 2 skirtingų puskanalių
12.	Jutiminis neuronas	Sensory neuron	
13.	Motorinis neuronas	Motor neuron	
14.	Vidaus neuronas	Interneuron	
15.	Atkarpomis tiesinių agregatų metodas	Piece Linear Aggregates (PLA)	Formalus matematinis specifikavimo metodas
16.	Paprastos diferencialinės lygtys	Ordinary differential equation (ODE)	
17.	Kvantinio lygio būsenų sistema	Quantized State System (QSS)	
18.	Pagrindinių komponenčių analizės metodas		Apima matematines procedūras, skirtas transformuoti tam tikrą kiekį galimai koreliuotų kintamųjų į

			mažesnį kiekį nekoreliuotų kintamųjų, vadinamų pagrindinėmis komponentėmis
19.	Tiesinės diskriminantinės analizės metodas		Metodas naudojamas statistikoje ir automatinio apsimokymo sistemose, skirtas surasti tiesinių požymių kombinaciją, kurie geriausiai atskiria dvi ar daugiau įvykių ar objektų klases
20.	Projekcijos paieškos metodas		Tai matematinės statistikos metodika, apimanti „labiausiai dominančios“ galimos projekcijos paiešką daugiamatėje duomenų struktūroje
21.	Daugiamačių skalių metodas		Tai aibė statistinių metodikų, dažniausiai naudojamų informacijos vaizdavime tiriant panašumus ir skirtumus duomenų struktūrose

3. SRITIES ANALIZĖ

3.1. Plyšinės jungties aprašymas

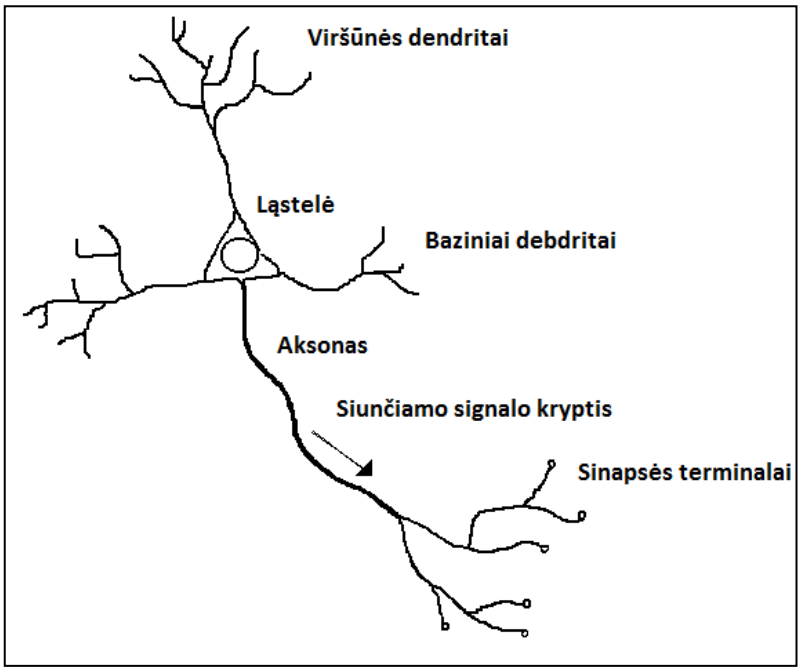
Įvairios medicininės laboratorijos, tiek mokymo įstaigų, tiek kitos (farmacijos ir pan.) atlieka specializuotus biologinius tyrimus. Viena tokių tyrimų sritis ląstelės elgsenos stebėjimas, bandymai suprasti bei aprašyti jų sandarą, struktūrą. Tuo tikslu turimas analizės objektas yra ląstelė ir jungtis tarp dviejų neuronų (vadinama plyšine jungtimi). Žinant kaip elgiasi mažiausias sistemos elementas galima nustatyti daugelio tokių elementų elgseną.

Daugelyje stuburinių audinių jonų ir mažų molekulių pasikeitimą tarp gretimų ląstelių valdo plyšinės jungtys, tokiu būdu koordinuojančios ląstelių veiklą. Plyšinė jungtis tiekia išteklius (jonus, mažas molekules), kad valdytų ląstelių veiklą audiniuose. Struktūriniu požiūriu plyšinė jungtis sudaryta iš koneksinų. Tai didelė membranos baltymų šeima, kurios pagalba tiesiogiai keičiamasi elektriniais ir metaboliniais signalais tarp kaimyninių ląstelių. Plyšinės jungtys leidžia sujungtoms ląstelėms perduoti elektrinius ir metabolinius signalus [1, 2, 3].

Panagrinėkime neuroną ir jo sandarą. Neuronai yra kelių rūšių:

- Jutiminis – pasižymi ilgu dendritu ir trumpu aksonu. „Žinutė“ perduodama iš daviklio į centrinę nervų sistemą.
- Motorinis – pasižymi ilgu aksonu ir trumpu dendritu. „Žinutė“ perduodama iš centrinės nervų sistemos į raumeninį audinį.
- Vidaus – aptinkamas tik centrinėje nervų sistemoje. Pagrindinė paskirtis sujungti neuronus tarpusavyje.

Neurone elektrinis signalas pasiduoda į viršūnės dendritus ir aksonu keliauja į kitą ląstelę ar raumenį jį sužadindamas. Neuroną sudaro: viršūnės dendritai; baziniai dendritai; ląstelė; aksonas; sinapsės terminalas. Paveiksle (Pav. 1) pavaizduotas motorinis neuronas.

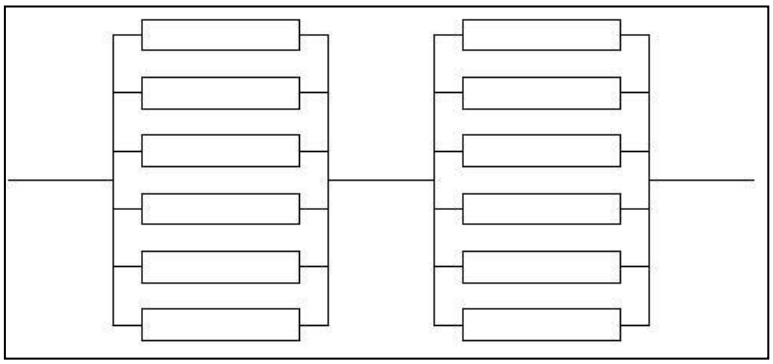


Pav. 1 Biologinio motorinio neurono sandara

Kai neuronas paveikiamas aplinkos, jis kitiems neuronams perduoda „pranešimą“ tuo sukeliant įtampos šuolį. Kiekviena ląstelė yra apsupta plonyte membrana, kuri atsakinga už cheminių medžiagų perdavimą ląstelei. Ląstelėse ir jos aplinkoje yra gausybė jonų (skirtingi kiekiai), kurie sukelia elektrochemines jėgas ląstelės membranoje. Membrana yra dalinai laidi, todėl vyksta jonų mainai ir susidaro įtampa, veikianti membraną.

3.1.1. Plyšinės jungties struktūra

Kiekvienas GJ kanalas sudarytas iš dviejų puskanalių, kurie savo ruožtu yra sudaryti iš šešių koneksinų subvienetų (Pav. 2). Iki šiol yra klonuota 21 skirtingų koneksinų žmogaus izoformų [4, 5].



Pav. 2 Plyšinės jungties kanalo struktūra

Tarpląstelinėje sąveikoje dalyvaujančios plyšinės jungtys gali būti homotipinės (identiški koneksinų izotipai abiejuose puskanaliuose), heterotipinės (skirtingi koneksinų izotipai puskanaliuose) ir heteromerinės (bent vienoje ląstelėje jungtį sudaro skirtingi koneksinų izotipai).

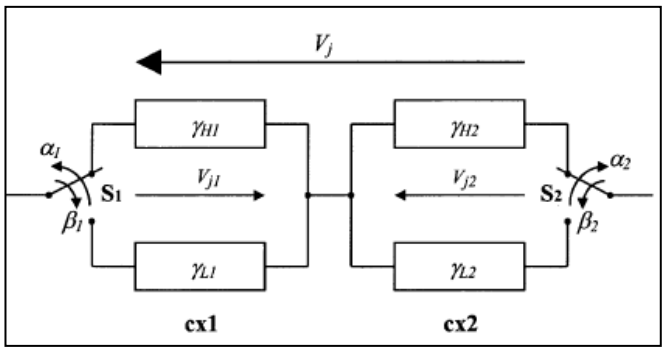
Plyšinės jungties kanalai skiriasi elektriniu laidumu, selektyviu pralaidumu, įvairioms cheminėms medžiagoms ir kanalų vartų reguliacija [2, 3].

Bendra plyšinės jungties kanalų savybė, kuri kaip manoma bendra visoms koneksinų izoformoms, yra kanalo vartų būsenos priklausomybė nuo įtampos [6, 7]. Jungties laidumas mažėja pridėjus tarpląstelinės jungties potencialą Vj. Nustatyta, kad kiekvienas puskanalis turi dviejų tipų vartų valdymo mechanizmus, skirtus jiems varstyti: lėtas, pilnai uždarantis plyšinę jungtį ir greitas, pervedantis vartus į pereinama būseną ar būsenas [8, 9]. Kiekvienas puskanalis gali būti dviejose būsenose: atviroje su laidumu GO ir uždarytoje su laidumu GC. Vartų kontrolės mechanizmai gali skirtis poliškumu (tai yra vartai gali būti uždaryti arba atidaryti), jeigu citoplazmos pusėje potencialas didėja arba mažėja. Jeigu abiejų puskanalių vartai turi ta pati poliškumą, tai su kiekvienu Vj nelygiu 0 vienas puskanalis atsidaro, o kitas užsidaro. Jeigu puskanalių vartai turi priešingą poliškumą, vienas Vj poliškumas atidarys abu kanalus, o priešingas Vj potencialas uždarys abu kanalus. Tokiu apibendrintu principu valdomas kanalas.

3.1.2. Plyšinės jungties kanalo konceptualusis modelis

Nustatyta, kad ląstelių tinklu sklinda elektrinis signalas, kadangi elektrinis signalas sklinda tinklu, tai galime teigti jog ir plyšine jungtimi signalas sklinda taip pat. Jei įvedame elektrinio signalo sąvoką, tai reiškia, jog galime aprašyti sistemą elektros grandinėmis.

Plyšinės jungties kanalo elektros grandinės schema atrodo taip:



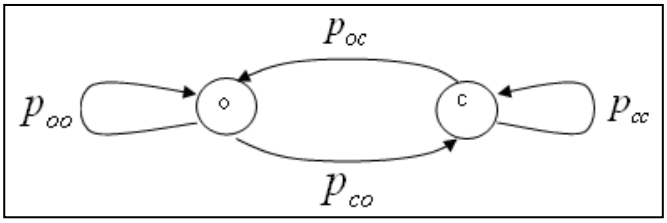
Pav. 3 Kanalas sudarytas iš 2 nuosekliai sujungtų vartų

Analizuojant Pav. 3 turime sąvoką „vartai“. Šie vartai leidžia arba neleidžia sklisti elektriniam signalui. Vartų būsena (vartai atidaryti arba uždaryti) apskaičiuojama remiantis tikimybinių dėsniu (yra išvestos specialios formulės). Taigi turime ryšį ***Medicina – elektros grandinių teorija – matematika (tikimybės)***. Informatikos vaidmuo šioje srityje toks , kad yra poreikis turėti galimybę imituoti biologinius procesus, atlikti eksperimentus neturint atitinkamos laboratorijos, registruoti atliktus stebėjimus, taigi reikalingas imitacinis modelis.

Toliau analizuojant plyšinę jungtį nustatyta, kad kanalas sudarytas iš dviejų nuosekliai sujungtų puskanalių. Puskanaliai yra nepriklausomi vienas nuo kito. Kiekvienas puskanalis yra sudarytas iš šešių koneksinų (subvartų), kurie gali būti arba atviroje (open, high), arba uždaroje

(closed, low, residual) būsenoje. Kiekvieno puskanalio koneksinas apibūdinamas laidumu g, kuris priklauso nuo pridedamos įtampos ir nuo būsenos, kuri keičiasi koneksinams reaguojant į įtampą. Gali būti keturi perėjimai tarp būsenų (žr. Pav. 4):

- pereis iš atviros (o - open) į uždarą (c - closed);
- pasiliks toje pačioje būsenoje (o) (vadinasi, pereis į open);
- pereis iš uždaros (c) į atvirą (o);
- pasiliks toje pačioje būsenoje (c).



Pav. 4 Perėjimai tarp būsenų

Vadinasi, ir perėjimų tikimybės gali būti keturios. Perėjimai iš vienos būsenos į kitą vyksta atsitiktinai.

Šis modelis yra stochastinis ir modelio parametrai (įtampa, laidis, srovės stipris yra priklausomi nuo laiko).

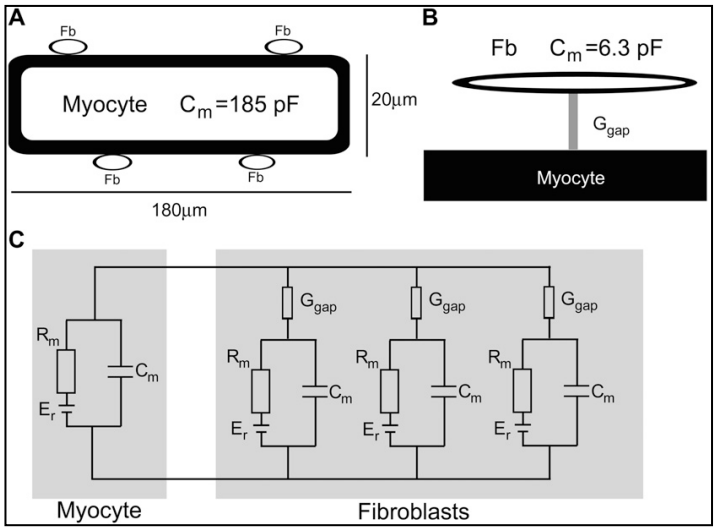
3.2. Plyšinės jungties vaidmuo ląstelių tinkluose

3.2.1. Širdies ląstelių tinkle

Miocitai ir fibroblastai yra dvi pagrindinės ląstelių rūšys sveiko žinduolio širdies skilvelio miokarde. Nors miocitai yra pagrinde atsakingi už mechanines funkcijas ir užima didžiąją audinių dalį, fibroblastai juos pranoksta savo kiekiu santykiu ~1,4 su 1 [10]. Šunų kairiajame skilvelyje fibroblastų buvo nustatyta esant tiesiogiai šalia kiekvieno miocito [11]. Be to plyšinės jungties proteinai, koneksinai yra išreikšti kario fibroblastais ir ląstelių junginiais iš fibroblastų į prieširdžio ir skilvelio miocitus. Tai leidžia manyti, kad miocitai ir fibroblastai yra bendrai veikianti pora jungiama plyšinės jungties ir bendraujanti abipusiais elektriniais signalais. Taip pat pastebėta, kad fibroblastai turi savybę sustiprinti nusilpusį elektrinį signalą. [12]

Eksperimentinėje sistemoje, sudarytoje iš dirbtinai išaugintų kardiomiocitų ir fibroblastų, erdvėje apibrėžtomis poromis, kur miocitai atskirti fibroblastais. Šis darinys parodė, kad bendravimas vyksta elektriniais signalais. Fibroblastai, kurie buvo suporuoti su miocitais atskleidė pokyčius membranos potenciale ir tai atitiko miocitų veikimo potencialą. Neseniai šie atradimai ir principai buvo praplėsti svarbiomis patofiziologijos principais. Taip pat buvo ištirta, kad dirbtinai išaugintų skirtingų miofibroblastų kiekių ant **miocitų** sluoksnių stebėjimų duomenys, gauti iš miocitų, parodė: santykis tarp veikimo potencialo depoliarizacijos ir laidumo greičio, mažėjo didėjant miofibroblastų tankui. [13,14,15]

Remiantis pastarųjų tyrimų duomenimis buvo sudarytas Tusscher modelis, kuris aprašo žmogaus širdies skilvelio veikimo potencialą. Schematiškai modelis pavaizduotas žemiau esančiame paveiksle (žr. Pav. 5).

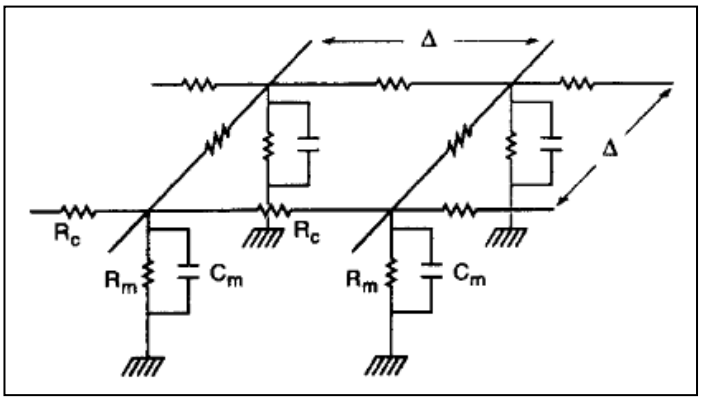


Pav. 5 Širdies ląstelių modelis

Brėžinyje pavaizduoti pagrindiniai matematinio modelio elektrinių sąveikų tarp skilvelio miocitų ir fibroblastų principai. A – bendra hibridinė sistema: vienas žmogaus skilvelio miocitas susietas tarpląstelinių laidumu su parinktu skaičiumi fibroblastu. B – miocito-fibroblasto poros supaprastintas grafinis vaizdas. Parodytas membranos talpumas (C_m) tarp miocito ir fibroblasto, kuriuos jungia plyšinė jungtis. C – miocitų ir fibroblastų elektrinės savybės atvaizduotos elektrine grandine. Kiekvienoje grandinės dalyje R_m apibūdina atitinkamos ląstelės membranos varžą, C_m – ląstelės elektrinė talpa, E_r – elektrovaros jėga ir G_{gap} – plyšinės jungties tarp miocito ir fibroblasto ląstelių varžą. [13, 16]

3.2.2. Bendra ląstelių ir plyšinių jungčių struktūra

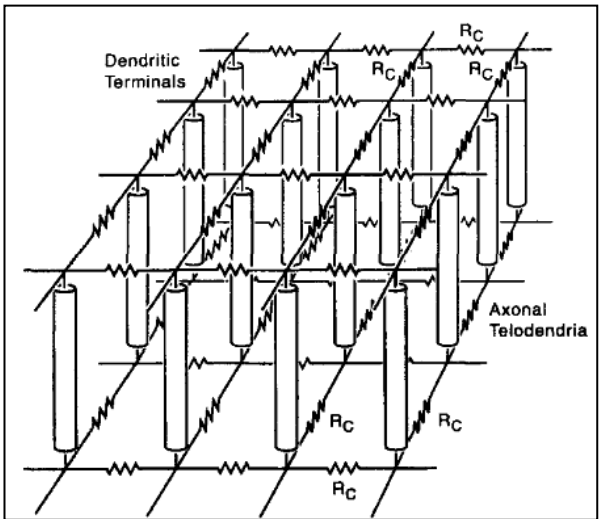
Supaprastintai ląstelių tinklo elektrines savybes galime aprašyti atvaizduodami ląsteles ir plyšines jungtis kaip elektros grandines dvimatėje erdvėje. Pvz.:



Pav. 6 Ląstelė dvimatėje erdvėje

Šiame brėžinyje vaizduojamas Lamb ir Simon dvidimensinis kvadratinis tinklo modelis [17], kuriame kiekviena tinklo dalis apibūdinamos atitinkamais grandinės komponentais. R_c – tai plyšinės jungties varža tarp dviejų gretimų ląstelių. Šiuo atveju ląstelę atitinka kvadrato viršūnė. Pati ląstelė aprašoma vertikaliai žemyn pavaizduota schema, kur C_m yra ląstelės elektrinė talpa, o R_m – ląstelės elektrinė varža.

Pasinaudodami ankščiau aprašytu modeliu galime pavaizduoti bet kokių ląstelių tinklus. Kaip pavyzdį pateikiame nervinių ląstelių tinklo modelį atvaizduotą trimatėje erdvėje, kurio pagrindui panaudotas Lamb ir Simon dvidimensinis kvadratinis tinklo modelis (Pav. 7).



Pav. 7 Trimatis ląstelės modelis remiantis Lamb ir Simon

Tarpląstelinių elektrinių jungčių būvimas tarp širdies ląstelių naudojant plyšinę jungtį yra būtinas išlaikant elektrinę tarpusavio sąveiką tam, kad palaikyti širdies ritmą, veiklos potencialo sklidimą ir koordinuoti depoliarizacijos procesą. Centrinėse širdies ląstelėse plyšinė jungtis yra reta ir akivaizdžiai atsitiktiniu būdu pasiskirsčiusi.

Atliekant ląstelių jungimo tyrimus dažniausiai buvo matuojamos pasyviosios elektrinės savybės, įvesties varža ir erdvės konstanta elektrotoniniam silpnėjimui. Įvertinant šiuos parametrus ir atsižvelgiant į ląstelių sujungimo teorinius modelius atsirado poreikis imituoti pasyviąsias elektrines savybes ir kuriomis remiantis, kaip įmanoma geriau, aprašyti miokardų audinį.

Tirti pasyvias elektrines savybes įvairių širdies dalių yra svarbu tam, kad būtų galima analizuoti sužadinimo sklidimą, aritmijos pradžios mechanizmus. Kadangi miokardas yra sudėtingas trimatis anizotropinis audinys (F. Bukauskas aprašo signalo sklidimą ląstelių tinkle bei modelių struktūras; pateikiamos formulės, skirtos skaičiuoti tokiam signalo sklidimui [21]), todėl neįmanoma tiesiogiai išmatuoti pasyviųjų elektrinių parametrų: tarpląstelinę kontaktų varžą, tarpląstelinę varžą išilgai ir skersai orientacijos ašies.

Vienas iš būdų matuoti pasyviąsias audinio su maža varža elektrines savybes ryšiuose tarp ląstelių, yra nustatyti elektrotoninę silpimo konstantą išilgai ir skersai ląstelės orientacijos ašių. Įsiurbimo elektrodas yra skirtas tiekti srovę tarpląstelinei terpei. Mikroelektrodas naudojamas matuoti elektrotoninio potencialo amplitudę skersai ir išilgai ląstelės orientacijos ašies. [21]

Plyšinė jungtis, jungianti kardiomiocitus yra priklausoma nuo įtampos ir laiko. Tokio tipo imitaciniu modeliu tiriamas dinaminės plyšinės jungties poveikis sklindančio veiklos potencialo laidžio kitimo greičiui. Pastarasis dinaminės plyšinės jungties modelis yra paremtas Vogel ir Wingart įtampos ir laiko priklausomybės modeliu, kuris nusako laidžio pokyčius ląstelių porose. Šiuo modeliu daroma prielaida, kad plyšinės jungties laidumo kanalai turi keturias struktūrines būsenas. Tai buvo panaudota sujungiant 300 ląstelių tiesia eilute, kai ląstelės aprašomos Luo-Rudy modeliu. Rezultatai parodė, kad ląstelėm esant stipriai suporuotom (didelis kanalų kiekis; apie ~6700 kanalų) mažai keičiasi plyšinės jungties varža vykstant signalo sklidimui. Taip pat pastebėti nežymūs pakitimai bangos formoje ir sklidimo greityje, kuomet lyginami statinių ir dinaminių plyšinių jungčių parametrai. Tuo tarpu sklindant signalui ir esant silpnam sujungimui (85 kanalai) plyšinės jungties varža smarkiai padidėja. Šis laikinas varžos pokytis padidino tarpjungtinio (*angl. transjunctional*) laidžio užlaikymus, pakitimus veiklos potencialo kilime. Remiantis šiais rezultatais daroma išvada, kad plyšinės jungties dinamines savybės padeda atskirti ląsteles naudojant tai kaip galimą apsaugos mechanizmą, t.y. esant reikalui izoliuoti pažeistas ląsteles nuo jų kaimynų.

Širdies audinyje elektriniai impulsai sklinda kaip vietinių grandinių srovės. Vietinės grandinės atitinka uždaras kilpas, apibūdinamas keturiais diskrečių struktūrų elementais:

1. Membranos kanalai, kurie neša sužadinančiąją vidinę srovę;
2. Citoplazma ir plyšinė jungtis paeiliui atvaizduoja tarpląstelinės varžos kelią;
3. Membranos talpinė varža sudaro sąlygas išoriniams pakeitimams;
4. Tarpląstelinis skystis veikia kaip varža.

Iš esmės šios keturios sąlygos apibrėžia potencialo veiklos kitimo greitį ir sudaro sąlygas laidžio susilpnėjimui. Tai imitavimo modelis, kuriame fiksuota plyšinės jungties varža pakeičiama dinamine tam, kad būtų galima nustatyti dinamines ryšio savybes. [22]

4. TYRIMAS

Galime išskirti kelias priežastis dėl kurių atsiranda poreikis turėti imitacinį modelį:

- Eksperimentai reikalauja atitinkamos laboratorijos, specialios įrangos, laiko. Tai yra ganėtinai brangu ir tokius tyrimus gali atlikti ribotas skaičius specialistų.
- Turimas imitacinis modelis leidžia pakartotinai atlikti eksperimentus norimu laiku ir vienodomis (eksperimentinėmis) sąlygomis, eksperimentų rezultatai pateikiami suprantamoje ir paprastoje formoje. Galimybė išsaugoti atliktą eksperimentą, bei vaizdžiai peržiūrėti ankstesnius rezultatus. Padidinamas specialistų skaičius, galintis dirbti šioje srityje, nes laboratorijos poreikio svarba ženkliai sumažėja.
- Imitacinio modelio pagalba galima gauti ženkliai daugiau tyrimo duomenų per trumpesnį laiką.

Panašaus pobūdžio programų jau yra sukurta, tačiau kiekvieno tyrimo specifika ir kryptis skiriasi, tai reiškia, kad viena universali sistema nėra tinkama visais atvejais, kadangi yra reikalingas skirtingas duomenų ir rezultatų tikslumas, tam tikri komponentai detalizuojami labiau ir pan. Esant tokioms aplinkybėms, tikslinga sukurti sistemą skirtą atlikti tyrimus konkrečioje probleminėje srityje, todėl analogiškos sistemos tinka tik pažvelgti kaip panašūs komponentai turėtų atrodyti bei veikti.

4.1. PLA specifikacija

Projektuojant sudėtingas sistemas padaroma daug klaidų. Pradiniame etape padarytos klaidos gali būti pastebėtos tik realizavus sistemą. Tai gali iššaukti daug problemų ir nuostolių. Siekiant to išvengti galima naudoti formalius metodus ir taip minimizuoti galimas klaidas ankstyvajame projekto etape. Formalūs metodai skaidomi į dvi stambias rūšis:

- Laiko diskretizavimo metodas;
- Įvykių diskretizavimo metodas.

Pirmasis metodas remiasi laiko skaidymu į tam tikrus laiko momentus ir ties kiekvienu laiko momentu atliekamas instrukcijas. Antrasis metodas remiasi sistemos įvykių diskretizavimu, kai nustatoma kuris įvykis pirmas turi įvykti. Pastarasis metodas vadinamas atkarpomis tiesinių agregatų metodu (PLA) [23, 24].

Kadangi plyšinės jungtys yra aprašytos elektros grandinėmis, todėl reikalingas metodas galintis spręsti paprastąsias diferencialines lygtis (ODE), kai kiekviena sekanti funkcijos reikšmė išskaičiuojama pagal esamą buvusią ankstesnės iteracijos turimą funkcijos išvestinę. Šiam tikslui įgyvendinti pasirinktas PLA formalių specifikacijų metodas. [24]

Sudėtingų sistemų formalios specifikacijos nagrinėjamos dviem požiūriais:

- Elgsenos. Tiriamos galimos sistemos trajektorijos. Patikrinama ar specifikacija sudaryta teisingai.
- Funkcionavimo. Teisingumas tikrinamas įvairiais validavimo ir verifikavimo metodais.

Imitacinio modeliavimo metu įvertinamos šios charakteristikos:

- Eilių ilgiai;
- Signalų perdavimo laikai;
- Laukimo laikai;
- Įrenginių koeficientai ir kt.

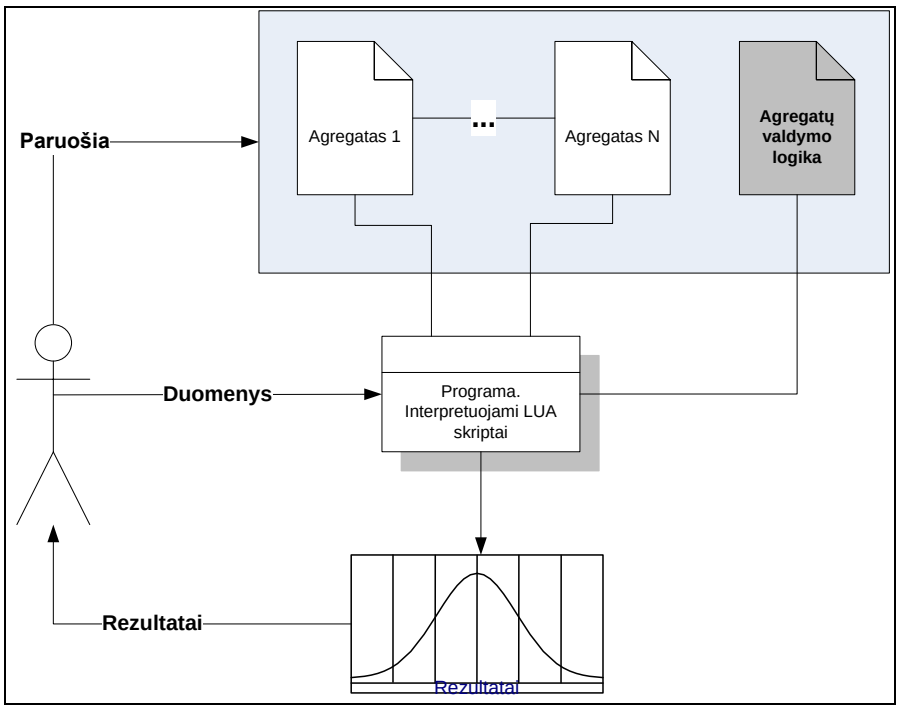
Pagrindinis formalių specifikacijų privalumas, kad sistema suprantama vienareikšmiškai, nes tai iš esmės matematinis metodas ir negali būti jokių dviprasmiškai interpretuojamų uždavinio elementų [24].

4.2. Modeliavimo programos architektūra

Elektrinio signalo sklidimui ląstelių tinkle imituoti buvo sukurta agregatinio modeliavimo programa. Realizuojamo modelio komponentai turi būti aprašyti LUA skriptų programavimo kalba. Agregatų tarpusavio sąveika, t.y. logika aprašoma taip pat LUA kalba. Programa, gebanti interpretuojanti LUA skriptus parašyta C++ kalba (programos autoriai Virginijus Pampikas ir Donatas Urbas). Tokios priemonės suteikia galimybę greitai ir nesudėtingai realizuoti modelio elementus, be to tam tikri agregatai (sudėtinės modelio dalys) gali būti pakartotinai panaudojami sprendžiant daugelį uždavinių (pvz.: integratoriai, įvairūs generatoriai ir pan.).

Tokia sistemos struktūra orientuota į agregatus, kurie gali būti formaliai aprašomi (pvz.: PLA specifikacijomis). Tokios specifikacijos leidžia vienareikšmiškai interpretuoti agregatų veikimą. Modelio realizacijos teisingumas priklauso nuo programuotojo kompetencijos.

Rašant skriptus privaloma realizuoti kelias sąsajos funkcijas, taip suteikiant programuotojui laisvę realizacijos požiūriu.

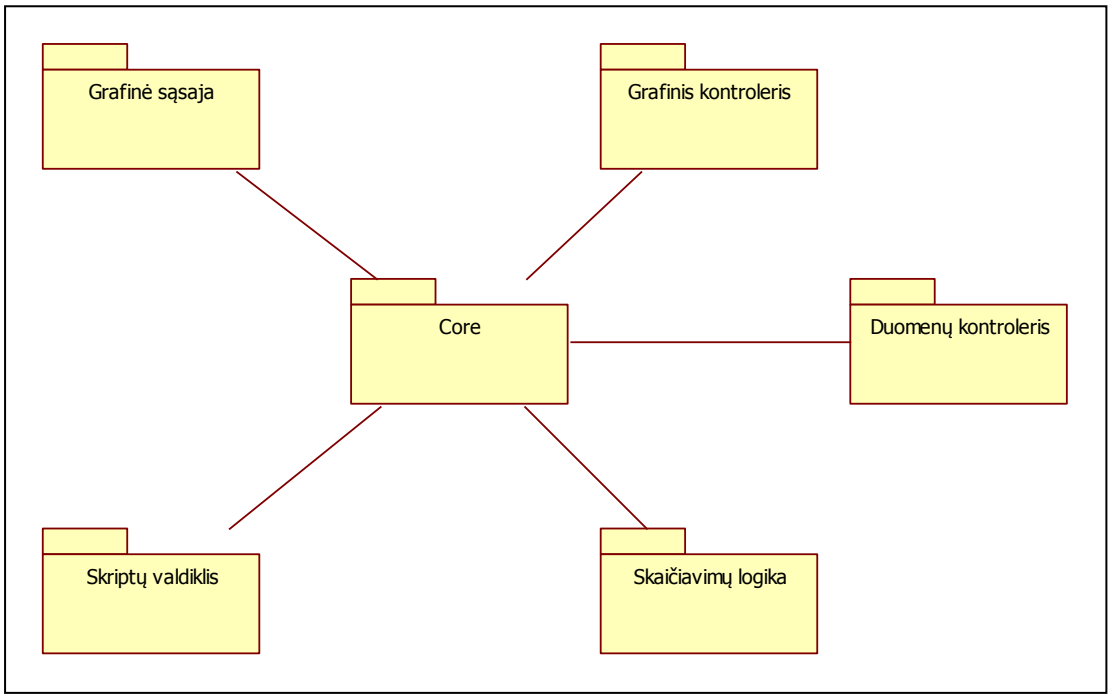


Pav. 8 Principinė naudojamos sistemos veikimo schema

Programą sudaro trys loginės dalys:

- Grafinė vartotojo sąsaja (GUI) - duomenų įvedimas ir rezultatų pateikimas;
- Skriptų interpretatorius - paruošto skripto kompiliavimas ir užkrovimas į programą;
- Skaičiavimų logika - imitacinių rezultatų skaičiavimas pagal pateiktus parametrus ir naudojamus skriptus.

Programa išskirtinė savo architektūra (programos paketų diagrama pavaizduota Pav. 9), ji pati konkrečių veiksmų susijusių su sprendžiamu uždaviniu neatlieka, tik interpretuoja LUA skriptus. Programa naudojama kaip įrankis imitatoriams kurti.



Pav. 9 Programos struktūra. Paketų diagrama

Paketų atsakomybės:

- Grafinė sąsaja. Paketas realizuoja grafinę vartotojo sąsają, t.y. jos komponentus. Aprašomi grafiniai objektai: mygtukai, tekstiniai įvesties komponentai, atskiri lango komponentai (slinkties juostos ir pan.), piešimo komponentai taip pat realizuoti ir įvykiai. Vartotojo sąsaja skaidoma į:
    - sistemos konfigūravimą;
    - duomenų įvedimą;
    - rezultatų pateikimą.
- Grafinis kontroleris. Šis paketas atsakingas už pastovų atvaizduojamų objektų sąrašo piešimą. Yra galimybė numatyti norimus atvaizdavimo būdus.
- Branduolys (Core). Paketas realizuoja duomenų struktūrų valdymą, perdavimą tam tikriems sistemos komponentams (pvz.: vaizduojamų objektų duomenų pakeitimas ar atnaujinimas).
- Skaičiavimų logika. Imitavimui reikalinga skaičiavimų logika aprašyta LUA kalbos skriptais ir skaidoma į dalis:
    - Atskirų tinklo elementų logika (ląstelė ir plyšinė jungtis);
    - Bendra signalo sklidimo tinkle logika.

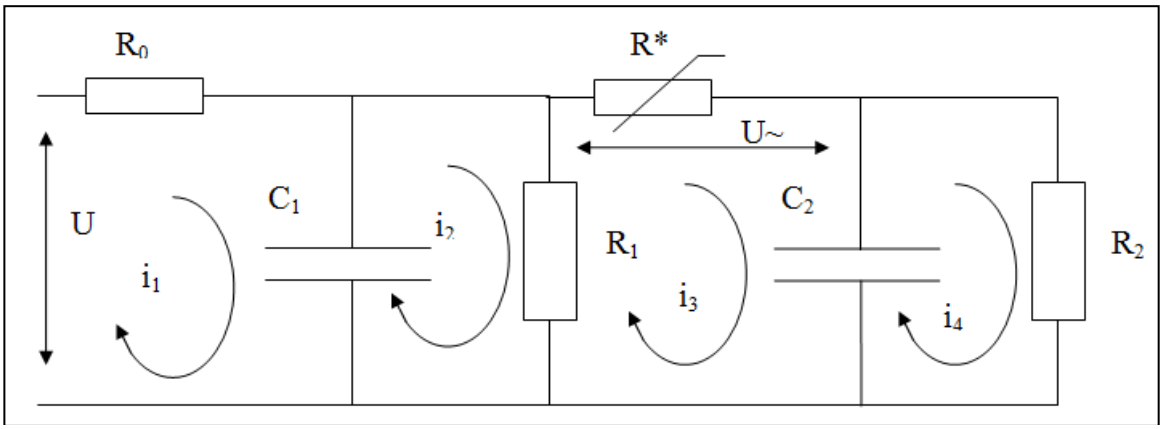
  Komponentai aprašyti kaip agregatai ir susieti su atitinkamais objektais. Vykdant skaičiavimus tinklo logikos skriptas per specialią sąsają kreipiasi į reikiamus objektus, kad paleisti vykdyti. Įvertinami atitinkamai parametrai, su kuriais siejasi skriptai.
- Skriptų valdiklis. Skriptais aprašytų veiksmų įkrovimas į programą bei paruošimas naudotis skriptuose aprašytomis funkcijomis.

Dalis sistemos paremta aprašytais skriptais, tai tokia sistemos specifika leidžia pakankamai lanksčiai keisti sistemą. Pernešamumas įtakoja tik reikiamo skripto perkopijavimą į kitą kompiuterį. Sistemos patikimumas priklauso nuo skriptuose realizuotų veiksmų teisingumo ir galima teigti jog už sistemos patikimumą, bei gaunamų rezultatų kokybę atsako skripto autorius.

Realizuotų agregatų bei jų valdymo logikos išeities tekstai pridedami priede.

4.3.Sprendimo metodas

Modeliuojant elektrinių signalų susijaudinimo perdavimą neuroninėmis ląstelėmis pasitelkiama elektros grandinių teoriją. Susijaudinimo signalą perduodamą per neuroninę ląstelę galime sutapatinti su elektriniu signalu, todėl tiek ląstelę, tiek ląstelių grandinę galime aprašyti elektrine grandine. Sudarytas tiriamojo signalo perdavimo kanalo modelis, kuris susideda iš dviejų ląstelių ir plyšinės jungties. Aprašytą modelį atitinka tokia elektros grandinių schema (žr. Pav. 10).



Pav. 10 Plyšinės jungties elektrinės grandinės schema

Kiekviena ląstelė yra sudaryta iš varžos R_n ir talpos C_n, o ląsteles jungia plyšinė jungtis (sinapsė) R^*. Plyšinė jungtis (sinapsė) jungia du neuronus ir ji yra pritaikyta susijaudinimo signalų perdavimui. Sinapsės yra dviejų rūšių – cheminės ir elektrinės. Plyšinė jungtis yra netiesinis elementas t.y. įtampos ir varžos (arba laidžio, kuris yra atvirkščias varžai) priklausomybė nėra tiesinė. Grandinei suteikiamas įtampos šuolis U, kuris simbolizuoja perduodamą signalą. Pradinio signalo stiprumą įtakoja pasyvioji varža R_0.

Elektrinės grandinės, pavaizduotos Pav. 10 lygtys aprašomos taip:

$$\begin{cases} \frac{di_1}{dt} = \frac{1}{R_0}\frac{du}{dt} - \frac{i_1 - i_2}{C_1 R_0} \\ \frac{di_2}{dt} = \frac{i_1 - i_2}{C_1}\left(\frac{1}{R^*} - \frac{1}{R_1}\right) - \frac{i_3 - i_4}{C_2 R^*} \\ \frac{di_3}{dt} = \frac{i_1 - i_2}{C_1 R^*} - \frac{i_3 - i_4}{C_2 R^*} \\ \frac{di_4}{dt} = \frac{i_3 - i_4}{C_2 R_2} \end{cases}$$

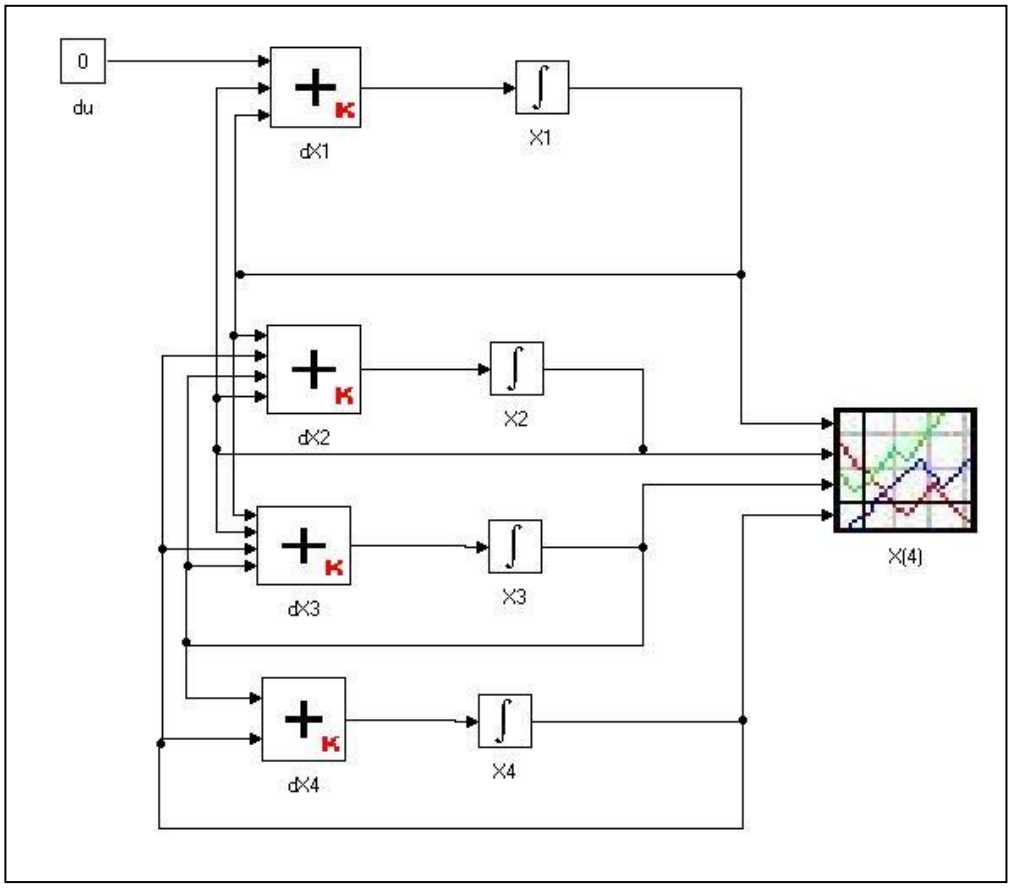
Iš šių lygčių sudaroma imitatoriaus schema (aprašyta 1.2.1 skyrelyje), atitinkanti pateiktą elektrinę grandinę ir sprendžianti šią grandinę išsprendžiančias lygtis.

4.3.1. Bendra imitatoriaus schema

Remiantis skyriuje sprendimo metodai ir algoritmai aprašytomis lygtimis sudarome imitatorių, kuris atitinka Pav. 10 pavaizduotą elektros grandinę. Imitatorius sudarytas iš:

- Išorinio signalo generatoriaus;
- Keturių sumatorių;
- Keturių integratorių.

Bendra imitatoriaus schema pavaizduota Pav. 11.



Pav. 11 Imitatoriaus schema

Toliau pateiktos formalios specifikacijos kiekvienam elementui atskirai.

4.3.2. Įtampos generatorius

1. Įėjimo signalų aibė $X = \varnothing$ - įtampa tarp elektros grandinės galų;
2. Išėjimo signalų aibė $Y = du(t_m)$ - įtampos U šuolis ($du(t_m) = \frac{dU}{dt}$);
3. Išorinių įvykių aibė $E' = \varnothing$;
4. Vidinių įvykių aibė $E'' = \{e_1''\}$;
   čia: e_1'' – įtampos pokytis tarp grandinės galų;

5. Diskrečioji agregato būsenos dedamoji $v(t_m)=\{du(t_m)\}$,
   čia $du(t_m)\in R$ – įtampos U pokytis;
6. Tolydžioji būsenos dedamoji $z_v(t_m)=\infty$;
7. Valdymo sekos $e_1''\mapsto\{\sigma_1\}$,
   čia: σ_1 – yra atsitiktinis įtampos šuolis kas 10 sekundžių,
   $$\sigma_1=\begin{cases}\lfloor rnd(10)\rfloor & kai\, 10\cdot i\le t_m<10\cdot i+1 \quad i=\{1,2,3,...1000\}\\ 0 & kitaip\end{cases}$$
   Pastaba: Operatorius $\lfloor z\rfloor$ – skaičiaus *z* sveikoji dalis, pvz. $\lfloor 6.123\rfloor=6$,
   $rnd(10)$ - atsitiktinis skaičius nuo 0 iki 10;
8. Pradinė būsena $v(t_0)=\{0\}$,
   $z_v(t_0)=\infty$;
9. Perėjimo ir išėjimo operatoriai:
   $H(e_1''):$ /* pasiekta įtampos reikšmė */
   $du(t_{m+1})=\sigma_1$
   $z(t_{m+1})=\infty$
   $G(e_1''):$
   $y=\sigma_1$

4.3.3. Sumatorius S1

1. Įėjimo signalų aibė $X=\{X_i, du\}$ – skaičiuojamos sumos dėmenys,
   čia $X_i\in R, i\in\{1,2\}$ – elektros srovė *i*-ojoje sekcijoje (iš **X1, X2 integratorių**),
   čia $du\in R$ – įtampos šuolis (iš **du**, t.y. įtampa tarp elektros grandinės galų);
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 $E'=\{e_1', e_2'\}$;
   čia: e_1 – pasikeitęs sumos operandas (t.y., pasikeitė išvestinės reikšmė),
   čia: e_2 – vienetinis sumos operandas (t.y., įvyko įtampos šuolis);
4. Vidinių įvykių aibė $E''=\varnothing$;
5. Diskrečioji agregato būsenos dedamoji
   $$v(t_m)=\left\{X_1(t_m), X_2(t_m), du, \beta_1=\frac{1}{C_1R_0}, \beta_2=\frac{1}{R_0}, S_1(t_m)\right\},$$
   čia $X_i(t_m)\in R,\quad i=\{1,2\}$ – sumos operandai (t.y. elektros srovė *i*-ojoje sekcijoje),
   $du\in R$ – sumos operandas (t.y. elektros įtampos šuolio greitis),
   $\beta_i(t_m)\in R,\quad i=\{1,2\}$ – 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: $v(t_m)=\{X_1(t_0), X_2(t_0), du, \beta_1, \beta_2, S_1(t_0)\}=\left\{0,0,0,\frac{1}{C_1R_0},\frac{1}{R_0},0\right\}$,
   $z_v(t_0)=\infty$;

9. Perėjimo ir išėjimo operatoriai:

$H(e_1'(x_k))$:

$X_k(t_{m+1}) = x_k$

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

$S_1(t_{m+1}) = X_2^* \cdot \beta_1 + X_1^* \cdot (-\beta_1)$

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

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

$G(e_1'')$:

$Y = X_2^* \cdot \beta_1 + X_1^* \cdot (-\beta_1)$

$H(e_2'(du))$:

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

$S_1(t_{m+1}) = X_2^* \cdot \beta_1 + X_1^* \cdot (-\beta_1) + du \cdot \beta_2$

čia $X_i^* = \{X_i(t_m), \quad i = 1,2$

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

$G(e_2'')$:

$Y = X_2^* \cdot \beta_1 + X_1^* \cdot (-\beta_1) + du \cdot \beta_2$

4.3.4. Sumatorius S2

1. Įėjimo signalų aibė $X = \{X_i\}$ – skaičiuojamos sumos dėmenys,
   čia $X_i \in R, i \in \{1,2,3,4\}$ – elektros srovė *i*-ojoje sekcijoje (iš **X1, X2, X3, X4 integratorių**);
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

$$v(t_m) = \left\{ X_1(t_m), X_2(t_m), X_3(t_m), X_4(t_m), \beta_1 = \frac{1}{C_1}\left(\frac{1}{R^*} - \frac{1}{R_1}\right), \beta_2 = \frac{1}{C_2 R^*}, S_2(t_m) \right\},$$

   čia $X_i(t_m) \in R, \quad i = \{1,2,3,4\}$ – sumos operandai (t.y. elektros srovė 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:

$$v(t_m) = \{X_1(t_0), X_2(t_0), X_3(t_0), X_4(t_0), \beta_1, \beta_2, S_2(t_0)\} = \left\{ 0,0,0,0, \frac{1}{C_1}\left(\frac{1}{R^*} - \frac{1}{R_1}\right), \frac{1}{C_2 R^*}, 0 \right\},$$

   $z_v(t_0) = \infty$;

9. Perėjimo ir išėjimo operatoriai:

$H(e_1'(x_k))$:

$X_k(t_{m+1}) = x_k$

$X_i(t_{m+1}) = X_i(t_m), \quad 1 \le i \le 4, \quad i \ne k$

$S_2(t_{m+1}) = X_1^* \cdot \beta_1 + X_2^* \cdot (-\beta_1) + X_3^* \cdot (-\beta_2) + X_4^* \cdot \beta_2$

čia $X_i^* = \begin{cases} X_i(t_m), & i \ne k \\ x_k, & i = k \end{cases} \quad i = 1,2,3,4$

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

$G(e_1'')$:

$Y = X_1^* \cdot \beta_1 + X_2^* \cdot (-\beta_1) + X_3^* \cdot (-\beta_2) + X_4^* \cdot \beta_2$

4.3.5. Sumatorius S3

1. Įėjimo signalų aibė $X = \{X_i\}$ – skaičiuojamos sumos dėmenys,
   čia $X_i \in R, i \in \{1,2,3,4\}$ – elektros srovė *i*-ojoje sekcijoje (iš **X1, X2, X3, X4 integratorių**);
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 reiškmė);
4. Vidinių įvykių aibė $E'' = \varnothing$;
5. Diskrečioji agregato būsenos dedamoji

   $v(t_m) = \left\{ X_1(t_m), X_2(t_m), X_3(t_m), X_4(t_m), \beta_1 = \frac{1}{C_1 R^*}, \beta_2 = \frac{1}{C_2 R^*}, S_3(t_m) \right\}$,

   čia $X_i(t_m) \in R, \quad i = \{1,2,3,4\}$ – sumos operandai (t.y. elektros srovė i-ojoje sekcijoje),
   $\beta_i(t_m) \in R, \quad i = \{1,2\}$ – 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:

   $v(t_m) = \{X_1(t_0), X_2(t_0), X_3(t_0), X4(t_0), \beta_1, \beta_2, S_3(t_0)\} = \left\{ 0,0,0,0, \frac{1}{C_1 R^*}, \frac{1}{C_2 R^*}, 0 \right\}$,

   $z_v(t_0) = \infty$;
9. Perėjimo ir išėjimo operatoriai:

   $H(e_1'(x_k))$:

   $X_k(t_{m+1}) = x_k$

   $X_i(t_{m+1}) = X_i(t_m), \quad i = 1,2,3,4 \quad i \ne k$

   $S_3(t_{m+1}) = X_1^* \cdot \beta_1 + X_2^* \cdot (-\beta_1) + X_3^* \cdot (-\beta_2) + X_4^* \cdot \beta_2$

   čia $X_i^* = \begin{cases} X_i(t_m), & i \ne k \\ x_k, & i = k \end{cases} \quad i = 1,2,3,4$

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

$G(e_1''):$

$Y = X_1^* \cdot \beta_1 + X_2^* \cdot (-\beta_1) + X_3^* \cdot (-\beta_2) + X_4^* \cdot \beta_2$

4.3.6. Sumatorius S4

1. Įėjimo signalų aibė $X = \{X_i\}$ – skaičiuojamos sumos dėmenys,
   čia $X_i \in R, i \in \{3,4\}$– elektros srovė *i*-ojoje sekcijoje (iš **X3, X4 integratorių**) ;
2. Išėjimo signalų aibė $Y = S_4(t_m)$ – suskaičiuota suma (t.y., išvestinės reikšmė $S_4(t_m) = \frac{dX_4}{dt}$);
3. Išorinių įvykių aibė $E' = \{e_1'\}$,
   čia: e_1 – pasikeitęs sumos operandas (t.y., pasikeitė diferencialo reiškmė);
4. Vidinių įvykių aibė $E'' = \varnothing$;
5. Diskrečioji agregato būsenos dedamoji $v(t_m) = \left\{ X_3(t_m), X_4(t_m), \beta_1 = \frac{1}{C_1 R_2}, S_4(t_m) \right\}$,
   čia $X_i(t_m) \in R, \quad i = \{3,4\}$ – sumos operandai (t.y. elektros srovė i-ojoje sekcijoje),
   $\beta_i(t_m) \in R, \quad i = \{1\}$ – iš anksto nustatyti sumos operandų koeficientai,
   $S_4(t_m)$ – suskaičiuota suma;
6. Tolydžioji būsenos dedamoji $z_v(t_m) = \infty$;
7. Valdymo sekos – $\varnothing$;
8. Pradinė būsena: $v(t_m) = \{X_3(t_0), X4(t_0), \beta_1, S_4(t_0)\} = \left\{0,0, \frac{1}{C_1 R_2}, 0\right\}$,
   $z_v(t_0) = \infty$;
9. Perėjimo ir išėjimo operatoriai:
   $H(e_1'(x_k)):$
   $X_k(t_{m+1}) = x_k$
   $X_i(t_{m+1}) = X_i(t_m), \quad i = 3,4 \quad i \neq k$
   $S_3(t_{m+1}) = X_3^* \cdot \beta_1 + X_4^* \cdot (-\beta_1)$
   čia $X_i^* = \begin{cases} X_i(t_m), & i \neq k \\ x_k, & i = k \end{cases} \quad i = 3,4$
   $z_v(t_{m+1}) = \infty$
   $G(e_1''):$
   $Y = X_3^* \cdot \beta_1 + X_4^* \cdot (-\beta_1)$

4.3.7. Integratorius X1

1. Įėjimo signalų aibė $X = \{S_1(t_m)\}$,
   čia: $S_1(t_m) \in R$ – skaičiuojamos funkcijos išvestinė ($S_1(t_m) = \frac{dX_1}{dt}$ iš **dX1 sumatoriaus**);
2. Išėjimo signalų aibė $Y = Q_{j_1}(t_m), \; j = 1 \ldots r$ - funkcijos X_1 (elektros srovė pirmojoje sekcijoje) kvantuota reikšmė;
3. Išorinių įvykių aibė $E' = \{e_1'\}$,
   čia: e_1' – atėjo pasikeitusi išvestinės reikšmė;

4. Vidinių įvykių aibė $E'' = \{e_1''\}$,
   čia: e_1'' – funkcija X_1 (elektros srovė 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 $e_1' \mapsto \{\sigma_1\}$, $e_1'' \mapsto \{\sigma_2\}$
   čia:
   σ_1 – 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:
8. $$\sigma_1 = \begin{cases} \dfrac{Q_{j_1(t_m)+1} - (X_1(t_m) + (t_{m+1} - t_m) \cdot x_1'(t_m))}{S_1(t_m)} & kai \ S_1(t_m) > 0 \\ \dfrac{(X_1(t_m) + (t_{m+1} - t_m) \cdot x_1'(t_m)) - (Q_{j_1(t_m)-1})}{|S_1(t_m)|} & kai \ S_1(t_m) < 0 \\ \infty \quad kai \ S_1(t_m) = 0 \end{cases}$$
   ir
   $$\sigma_2 = \begin{cases} \dfrac{Q_{j_1(t_m)+2} - (X_1(t_m) + (t_{m+1} - t_m) \cdot x_1'(t_m))}{x_1'(t_m)} & kai \ x_1'(t_m) > 0 \\ \dfrac{(X_1(t_m) + (t_{m+1} - t_m) \cdot x_1'(t_m)) - (Q_{j_1(t_m)-1})}{|x_1'(t_m)|} & kai \ d_{x1}(t_m) < 0 \\ \infty \quad kai \ x_1'(t_m) = 0 \end{cases}$$
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 */
    $X_1(t_{m+1}) = X_1(t_m) + (t_{m+1} - t_m) \cdot x_1'(t_m)$
    $d_{x1}(t_{m+1}) = x_{1v}$
    $j_1(t_{m+1}) = j_1(t_m)$
    $w(e_1', t_{m+1}) = t_m + \sigma_1$
    $G(e_2')$: $y = Q_{j_1(t_m)} + \lfloor (t_{m+1} - t_m) \cdot x_1'(t_m) + x_y / \Delta Q \rfloor$

$H(e_1''):$ /* vidinis įvykis, nepakeistas */

$X_1(t_{m+1}) = X_1(t_m) + \sigma_1 \cdot x_1'(t_m)$

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

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

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

$G(e_1''):\quad y = Q_{j_1(t_m)+\mathrm{sgn}(x_1'(t_m))}$

4.3.8. Integratorius X2

1. Įėjimo signalų aibė $X = S_2(t_m) \in R$ – skaičiuojamos funkcijos X_2 išvestinė ($S_2(t_m) = \frac{dX_2}{dt}$ iš **dX2 sumatoriaus**);
2. Išėjimo signalų aibė $Y = Q_{j_2}(t_m),\ j = 1 \ldots r$ – funkcijos X_2 (elektros srovė antrojoje sekcijoje) kvantuota reikšmė;
3. Išorinių įvykių aibė $E' = \{e_1'\}$,
   čia: e_1' – atėjo nauja išvestinės reikšmė (iš **dX2 sumatoriaus**);
4. Vidinių įvykių aibė $E'' = \{e_1''\}$,
   čia: e_1'' – elektros srovė antrojoje sekcijoje pasiekė sekantį funkcijos kvantinį slenkstį;
5. Diskrečioji agregato būsenos dedamoji $v(t_m) = \{X_2(t_m), x_2'(t_m), j_2(t_m)\}$,
   čia $X_2(t_m) \in R$ – apskaičiuota funkcijos X_2 (elektros srovė antrojoje sekcijoje) reikšmė,
   $x_2'(t_m) \in R$ – esama funkcijos išvestinės reikšmė,
   $j_2(t_m) \in Z$ – funkcijos X_2 kvantuotos būsenos numeris;
6. Tolydžioji būsenos dedamoji $z_v(t_m) = \{w(e_1'', t_m)\}$ – laiko momentas, kada funkcija X_2 pasieks naują kvantinį slenkstį,
   $$w(e_1'', t_m) = \begin{cases} < \infty, \ x_2'(t_m) \neq 0 \\ \infty \ \textit{priešingu atveju} \end{cases};$$
7. Valdymo sekos $e_1' \mapsto \{\sigma_1\},\ e_1'' \mapsto \{\sigma_2\}$,
   čia:
   σ_1 – yra laiko tarpas, per kurį funkcija X_2 pasieks sekančią kvantuotą reikšmę po išorinio įvykio,
   σ_2 – yra laiko tarpas, per kurį funkcija X_2 pasieks sekančią kvantuotą reikšmę po vidinio įvykio;
8. $$\sigma_1 = \begin{cases} \dfrac{Q_{j_2(t_m)+1} - (X_2(t_m) + (t_{m+1} - t_m) \cdot x_2'(t_m))}{S_2(t_m)} & kai\ S_2(t_m) > 0 \\ \dfrac{(X_2(t_m) + (t_{m+1} - t_m) \cdot x_2'(t_m)) - (Q_{j_2(t_m)-1})}{|S_2(t_m)|} & kai\ S_2(t_m) < 0 \\ \infty \quad kai\ S_2(t_m) = 0 \end{cases}$$
   ir

$$\sigma_2 = \begin{cases} \dfrac{Q_{j_2(t_m)+2} - \left(X_2(t_m) + (t_{m+1} - t_m)\cdot x_2'(t_m)\right)}{x_2'(t_m)} & kai\ x_2'(t_m) > 0 \\ \dfrac{\left(X_2(t_m) + (t_{m+1} - t_m)\cdot x_2'(t_m)\right) - \left(Q_{j_2(t_m)-1}\right)}{\left|x_2'(t_m)\right|} & kai\ x_2'(t_m) < 0 \\ \infty \quad kai\ x_2'(t_m) = 0 \end{cases}$$

kai:

$Q_1, Q_2, \ldots Q_r$ – funkcijos diskretizavimo tinklelis,

$Q_k - Q_{k-1} = \Delta Q$ – funkcijos kvantas;

9. Pradinė būsena $v(t_0) = \{X_2(t_0), x_2'(t_0), j_2(f(X_2(t_0)))\}$,
   $z_v(t_0) = \{t_0 + \sigma_2\}$;
10. Perėjimo ir išėjimo operatoriai:
    $H(e_1'(S_2(t_m)))$: /* pasikeitė išvestinės reikšmė */
    $X_2(t_{m+1}) = X_2(t_m) + (t_{m+1} - t_m)\cdot x_2'(t_m)$
    $x_2'(t_{m+1}) = S_2(t_m)$
    $j_2(t_{m+1}) = j_2(t_m)$
    $w(e_1', t_{m+1}) = t_m + \sigma_1$
    $H(e_1'')$: /* pasiekta sekanti funkcijos reikšmė */
    $X_2(t_{m+1}) = X_2(t_m) + (t_{m+1} - t_m)\cdot x_2'(t_m)$
    $x_2'(t_{m+1}) = x_2'(t_m)$
    $j_2(t_{m+1}) = j_2(t_m) + \mathrm{sgn}(x_2'(t_m))$

    čia:

    $\mathrm{sgn}(v) = \begin{cases} 1 \quad kai\ v > 0 \\ -1\ kai\ v < 0 \end{cases}$, jei įvyko vidinis įvykis, vadinasi $v = x_2'(t_m) \neq 0$

    $w(e_1'', t_{m+1}) = t_m + \sigma_2$
    $G(e_1'')$:
    $y = Q_{j_2(t_m)+\mathrm{sgn}(x_2'(t_m))}$

4.3.9. Integratorius X3

1. Įėjimo signalų aibė $X = S_3(t_m) \in R$ – skaičiuojamos funkcijos išvestinė ($S_3(t_m) = \dfrac{dX_3}{dt}$ iš **dX3 sumatoriaus**);
2. Išėjimo signalų aibė $Y = Q_j(t_m)$, $j = 1 \ldots r$ - funkcijos X_3 kvantuota reikšmė;
3. Išorinių įvykių aibė $E' = \{e_1'\}$,
   čia: e_1' – atėjo nauja išvestinės reikšmė (iš **dX3 sumatoriaus**);
4. Vidinių įvykių aibė $E'' = \{e_1''\}$,
   čia: e_1'' – elektros srovė trečiojoje sekcijoje pasiekė sekantį funkcijos kvantinį slenkstį;
5. Diskrečioji agregato būsenos dedamoji $v(t_m) = \{X_3(t_m), x_3'(t_m), j_3(t_m)\}$,
   čia $X_3(t_m) \in R$ – apskaičiuota funkcijos X_3 (elektros srovė trečiojoje sekcijoje) reikšmė,
   $x_3'(t_m) \in R$ – diferencialo reikšmė,
   $j_3(t_m) \in Z$ – funkcijos X_3 kvantuotos būsenos numeris;

6. Tolydžioji būsenos dedamoji $z_v(t_m)=\{w(e_1'',t_m)\}$ – laiko momentas, kada funkcija X_3 pasieks naują kvantinį slenkstį,

$$w(e_1'',t_m)=\begin{cases} <\infty,\ x_3'(t_m)\neq 0 \\ \infty\ priešingu\ atveju \end{cases};$$

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

čia:

σ_1 – yra laiko tarpas, per kurį funkcija X_3 pasieks sekančią kvantuotą reikšmę po išorinio įvykio,

σ_2 – yra laiko tarpas, per kurį funkcija X_3 pasieks sekančią kvantuotą reikšmę po vidinio įvykio;

8.

$$\sigma_1=\begin{cases} \dfrac{Q_{j_3(t_m)+1}-(X_3(t_m)+(t_{m+1}-t_m)\cdot x_3'(t_m))}{S_3(t_m)} & kai\ S_3(t_m)>0 \\ \dfrac{(X_3(t_m)+(t_{m+1}-t_m)\cdot x_3'(t_m))-(Q_{j_3(t_m)-1}-\varepsilon)}{|S_3(t_m)|} & kai\ S_3(t_m)<0 \\ \infty \quad kai\ S_3(t_m)=0 \end{cases}$$

ir

$$\sigma_2=\begin{cases} \dfrac{Q_{j_3(t_m)+2}-(X_3(t_m)+(t_{m+1}-t_m)\cdot x_3'(t_m))}{x_3'(t_m)} & kai\ x_3'(t_m)>0 \\ \dfrac{(X_3(t_m)+(t_{m+1}-t_m)\cdot x_3'(t_m))-(Q_{j_3(t_m)-1})}{|x_3'(t_m)|} & kai\ x_3'(t_m)<0 \\ \infty \quad kai\ x_3'(t_m)=0 \end{cases}$$

kai:

$Q_1,Q_2,\ldots Q_r$ – funkcijos diskretizavimo tinklelis,

$Q_k-Q_{k-1}=\Delta Q$ – funkcijos kvantas;

9. Pradinė būsena $v(t_0)=\{X_3(t_0),x_3'(t_0),j_3(f(X_3(t_0)))\}$,

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

10. Perėjimo ir išėjimo operatoriai:

$H(e_1'(S_3(t_m)))$: /* pasikeitė išvestinės reikšmė */

$X_3(t_{m+1})=X_3(t_m)+(t_{m+1}-t_m)\cdot x_3'(t_m)$

$x_3'(t_{m+1})=S_3(t_m)$

$j_3(t_{m+1})=j_3(t_m)$

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

$H(e_1'')$: /* pasiekta sekanti funkcijos reikšmė */

$X_3(t_{m+1})=X_3(t_m)+(t_{m+1}-t_m)\cdot x_3'(t_m)$

$x_3'(t_{m+1})=x_3'(t_m)$

$j_3(t_{m+1})=j_3(t_m)+\mathrm{sgn}(x_3'(t_m))$

čia:

$\mathrm{sgn}(v)=\begin{cases} 1 \quad kai\ v>0 \\ -1\ kai\ v<0 \end{cases}$, jei įvyko vidinis įvykis, vadinasi $v = x_3'(t_m) \neq 0$

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

$G(e_1'')$:

$y = Q_{j_3(t_m)+\mathrm{sgn}(x_3'(t_m))}$

4.3.10. Integratorius X4

1. Įėjimo signalų aibė $X = S_4(t_m) \in R$ – skaičiuojamos funkcijos išvestinė ($S_4(t_m) = \frac{dX_4}{dt}$ iš **dX4 sumatoriaus**);
2. Išėjimo signalų aibė $Y = Q_j(t_m),\ j = 1 \ldots r$ – funkcijos X_4 kvantuota reikšmė;
3. Išorinių įvykių aibė $E' = \{e_1'\}$,
   čia: e_1' – atėjo nauja išvestinės reikšmė (iš **dX4 sumatoriaus**);
4. Vidinių įvykių aibė $E'' = \{e_1''\}$,
   čia: e_1'' – elektros srovė ketvirtojoje sekcijoje pasiekė sekantį funkcijos kvantinį slenkstį;
5. Diskrečioji agregato būsenos dedamoji $v(t_m) = \{X_4(t_m), x_4'(t_m), j_4(t_m)\}$,
   čia $X_4(t_m) \in R$ – apskaičiuota funkcijos X_4 (elektros srovė ketvirtojoje sekcijoje) reikšmė,
   $x_4'(t_m) \in R$ – diferencialo reikšmė,
   $j_4(t_m) \in Z$ – funkcijos X_4 kvantuotos būsenos numeris;
6. Tolydžioji būsenos dedamoji $z_v(t_m) = \{w(e_1'', t_m)\}$ – laiko momentas, kada funkcija X_4 pasieks naują kvantinį slenkstį,
   $$w(e_1'', t_m) = \begin{cases} < \infty,\ x_4'(t_m) \neq 0 \\ \infty\ priešingu\ atveju \end{cases};$$
7. Valdymo sekos $e_1' \mapsto \{\sigma_1\},\ e_1'' \mapsto \{\sigma_2\}$,
   čia: σ_1 – yra laiko tarpas, per kurį funkcija X_4 pasieks sekančią kvantuotą reikšmę po išorinio įvykio,
   σ_2 – yra laiko tarpas, per kurį funkcija X_4 pasieks sekančią kvantuotą reikšmę po vidinio įvykio;
8. $$\sigma_1 = \begin{cases} \dfrac{Q_{j_3(t_m)+1} - (X_4(t_m) + (t_{m+1} - t_m) \cdot x_4'(t_m))}{S_4(t_m)} & kai\ S_4(t_m) > 0 \\ \dfrac{(X_4(t_m) + (t_{m+1} - t_m) \cdot x_4'(t_m)) - (Q_{j_3(t_m)-1})}{|S_4(t_m)|} & kai\ S_4(t_m) < 0 \\ \infty \quad kai\ S_4(t_m) = 0 \end{cases}$$

   ir

   $$\sigma_2 = \begin{cases} \dfrac{Q_{j_3(t_m)+2} - (X_4(t_m) + (t_{m+1} - t_m) \cdot x_4'(t_m))}{x_4'(t_m)} & kai\ x_4'(t_m) > 0 \\ \dfrac{(X_4(t_m) + (t_{m+1} - t_m) \cdot x_4'(t_m)) - (Q_{j_3(t_m)-1})}{|x_4'(t_m)|} & kai\ x_4'(t_m) < 0 \\ \infty \quad kai\ x_4'(t_m) = 0 \end{cases}$$

   kai:

$Q_1, Q_2, \ldots Q_r$ – funkcijos diskretizavimo tinklelis,

$Q_k - Q_{k-1} = \Delta Q$ – funkcijos kvantas;

9. Pradinė būsena $v(t_0) = \{X_4(t_0), x_4'(t_0), j_4(f(X_4(t_0)))\}$,

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

10. Perėjimo ir išėjimo operatoriai:

$H(e_1'(S_4(t_m)))$: /* pasikeitė išvestinės reikšmė */

$X_4(t_{m+1}) = X_4(t_m) + (t_{m+1} - t_m) \cdot x_4'(t_m)$

$x_4'(t_{m+1}) = S_4(t_m)$

$j_4(t_{m+1}) = j_4(t_m)$

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

$H(e_1'')$: /* pasiekta sekanti funkcijos reikšmė */

$X_1(t_{m+1}) = X_4(t_m) + (t_{m+1} - t_m) \cdot x_4'(t_m)$

$x_4'(t_{m+1}) = x_4'(t_m)$

$j_4(t_{m+1}) = j_4(t_m) + \text{sgn}(x_4'(t_m))$

čia:

$\text{sgn}(v) = \begin{cases} 1 \quad kai\ v > 0 \\ -1\ kai\ v < 0 \end{cases}$, jei įvyko vidinis įvykis, vadinasi $v = x_4'(t_m) \neq 0$

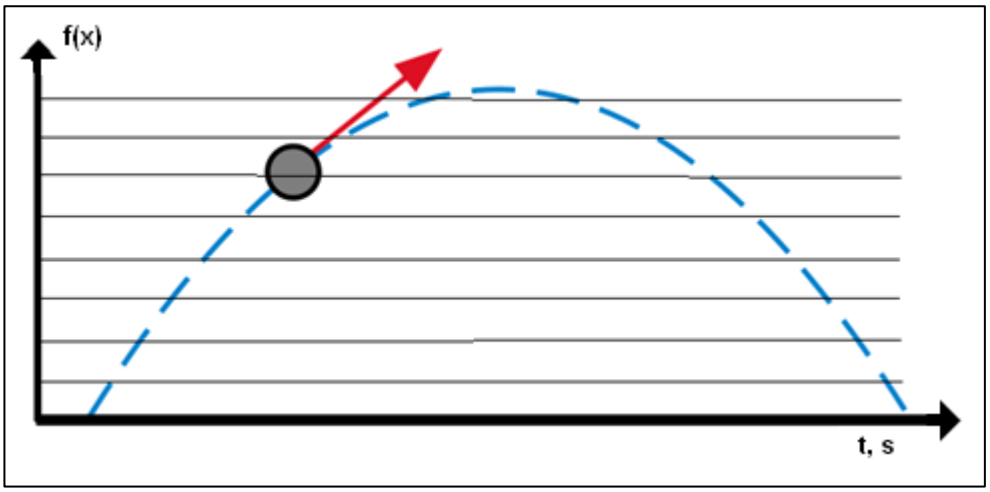
$w(e_1'', t_{m+1}) = t_m + \sigma_2$

$G(e_1'')$:

$y = Q_{j_4(t_m) + \text{sgn}(x_4'(t_m))}$

4.4. Sprendimo algoritmas

Pateiktas grubus ir supaprastintas algoritmo paaiškinimas. Elektros grandinės (žr. Pav. 10) sprendimas atliktas remiantis PLA siūloma metodika ir minėtai elektros grandinei sudarytomis formaliomis specifikacijomis. Funkcijos reikšmės suradimui įvedama sąvoka kvantinis lygis. Kvantinis lygis – tai konkreti funkcijos reikšmė. Turime dekarto koordinačių sistemą, kai Y ašies reikšmės išskaidytos į tam tikrą intervalų skaičių ir kiekvieną esamą intervalą atitinka tam tikra reikšmė, be to esamą ir sekantį intervalą atitinkančią reikšmę skiria vienodas dydis. Tokiu būdu diskretizuojamos galimos funkcijos reikšmės. Verta pažymėti, kad rezultatų tikslumas bei skaičiavimų apimtis priklauso nuo pasirinkto pokyčio dydžio tarp dviejų kvantinių lygių (žr. Pav. 12).



Pav. 12 Reikšmių skaidymas į kvantinius lygius

Imitatoriaus veiksmai pažingsniui (žr. Imitatoriaus schemą Pav. 11):

1. Pirmoje iteracijoje sugeneruojamas išorinio įvykio signalas (sužadinama sistema), kuris po to bus naujai paduodamas kas nustatytą laiko intervalą (pvz.: kas 0,7 sekundės). Išorinis signalas fiksuojamas kaip įvykis.
2. Suskaičiuoti naują kintamos varžos reikšmę ir ją perduoti reikiamiems sumatoriams.
3. Atliekami sumatorių veiksmai (skaičiuojama funkcijos išvestinės reikšmė), kurie aprašyti specifikacijose 4.3 skyriuje.
4. Reikšmės gautos iš sumatorių perduodamos į integratorius (skaičiavimai aprašyti specifikacijoje). Kiekvienas integratorius grąžina laiką kada bus pasiekta nauja kvantinio lygio reikšmė. Funkcijos grafiko kitimo greitį apsprendžia funkcijos išvestinė. Kiekviena integratoriaus išvestis yra laikoma kaip atskiras įvykis.
5. Surandamas minimalus laikas tarp įvykių. Gaunama viena elektros srovės kontūro reikšmė.
6. Vykdomi veiksmai nuo 2 punkto.

Pastaba: Kada kurie imitatoriaus elementai tiksliai yra iškviečiamas žiūrėti į schemą (Pav. 11).

4.5. Imitatoriaus dalių testavimas

Siekiant korektiškų plyšinių jungčių modelio darbo rezultatų būtina užtikrinti, kad kiekviena modelio dalis veikia korektiškai. Tam tikslui kiekvienam agregatui pagal atitinkamą specifikaciją buvo sukurta aibė testinių atvejų. Pagal agregato realizaciją paduodami duomenys ir laukiama specifikaciją atitinkančių rezultatų.

Testavime naudojamos elektros grandinės komponentų reikšmės (sumatorių daugiklių reikšmės žr. formalią specifikaciją 4.3.1 – 4.3.10 skyreliai).

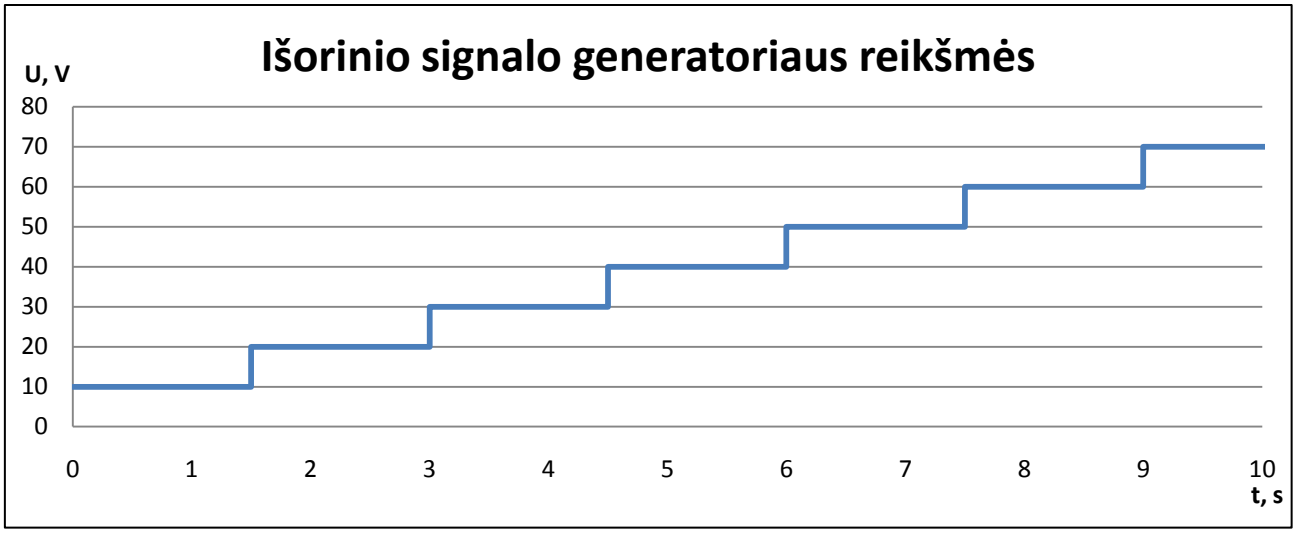
Elementas	**Reikšmė**
R_0	100 Ω
R_1	1000 Ω
R_2	1000 Ω
R*	Kintamos varžos atveju kinta nuo 0 Ω iki 25 Ω. Pastovios varžos

	atveju reikšmė 100 Ω.
C_1	0,001 F
C_2	0,001 F

4.5.1. Išorinio signalo generatoriaus testavimas

Išorinio signalo generatoriaus paskirtis kas nustatytą imitacijos laiko tarpą sugeneruoti įtampos reikšmę, kuri kiekvienu nauju reikiamu laiko momentu grąžintų pakeltą įtampos reikšmę nustatytu dydžiu. Jei tokius rezultatus atvaizduotume grafiškai, tai turime gauti „laiptus“.

Generatorius turi sugeneruoti reikšmes 10 sekundžių atkarpai kas 1.5 sekundės įtampos reikšmę pakeliant 10 V.



Pav. 13 Išorinio signalo sugeneruotos reikšmės

Pagal gautus rezultatus galime teigti, kad agregatas veikia korektiškai. Išbrėžtas grafikas atitinka „laiptų“ formą.

4.5.2. Sumatoriaus 1 testavimas

Sumatoriaus 1 atsakomybė gavus du išorinius signalus, juos padauginti iš koeficiento ir susumuoti. Testuojamas sumatoriaus įvykis, kai pasikeitė sumos operandas (funkcijos išvestinės reikšmė). Varža R* nekintama ir lygi 100. Koeficiento reikšmė $\beta_1 = 10$. Atliekamas veiksmas: Rezultatas = $X1 \cdot (-\beta_1) + X2 \cdot \beta_1$

Lentelė 2 Sumatoriaus 1 testavimo rezultatai

X1	X2	Laukiamas rezultatas	Agregato rezultatas	Pastaba
0	0	0	0	Abi įvestys 0
5	0	-50	-50	X1 > 0
0	5	50	50	X2 > 0
5	5	0	0	Abi įvestys > 0
-5	0	50	50	X1 < 0
0	-5	-50	-50	X2 < 0
-5	-5	0	0	X1, X2 < 0

Pagal lentelės rezultatus matome, kad sumatorius veikia tvarkingai ir nedaro klaidų.

4.5.3. Sumatoriaus 2 testavimas

Sumatoriaus 2 atsakomybė gavus keturis išorinius signalus, juos padauginti iš koeficientų ir susumuoti. Testuojamas sumatoriaus įvykis, kai pasikeitė sumos operandas (funkcijos išvestinės reikšmė). Varža R* nekintama ir lygi 100. Koeficiento reikšmė $\beta_1 = 9$, $\beta_2 = 10$. Atliekamas veiksmas: Rezultatas = $X1 \cdot \beta_1 + X2 \cdot (-\beta_1) + X3 \cdot (-\beta_2) + X4 \cdot \beta_2$. Čia parodomas sumatoriaus veikimas, kai įvestys yra teigiamos arba lygios 0.

Lentelė 3 Sumatoriaus 2 testavimo rezultatai

X1	X2	X3	X4	Laukiamas rezultatas	Agregato rezultatas
0	0	0	0	0	0
0	0	0	5	50	50
0	0	5	0	-50	-50
0	5	0	0	-45	-45
5	0	0	0	45	45
0	0	5	5	0	0
0	5	0	5	5	5
5	0	0	5	95	95
0	5	5	0	-95	-95
5	5	0	0	0	0
5	5	5	0	-50	-50
5	0	5	5	45	45
5	5	0	5	50	50
0	5	5	5	-45	-45
5	5	5	5	0	0

Pagal lentelės rezultatus matome, kad sumatorius veikia tvarkingai ir nedaro klaidų.

4.5.4. Sumatoriaus 3 testavimas

Testuojant sumatorių 3 įvesčių skaičius sutampa su sumatoriaus 2 įvesčių skaičiumi, todėl bus panaudota ta pati įvesčių aibė. $\beta_1 = 9$, $\beta_2 = 10$. Atliekamas veiksmas: Rezultatas = $X1 \cdot \beta_1 + X2 \cdot (-\beta_1) + X3 \cdot (-\beta_2) + X4 \cdot \beta_2$

Lentelė 4 Sumatoriaus 3 testavimo rezultatai

X1	X2	X3	X4	Laukiamas rezultatas	Agregato rezultatas
0	0	0	0	0	0
0	0	0	5	50	50
0	0	5	0	-50	-50
0	5	0	0	-50	-50
5	0	0	0	50	50
0	0	5	5	0	0
0	5	0	5	0	0
5	0	0	5	100	100
0	5	5	0	-100	-100
5	5	0	0	0	0

5	5	5	0	-50	-50
5	0	5	5	50	50
5	5	0	5	50	50
0	5	5	5	-50	-50
5	5	5	5	0	0

Pagal lentelės rezultatus matome, kad sumatorius veikia tvarkingai ir nedaro klaidų.

4.5.5. Sumatoriaus 4 testavimas

Sumatoriaus 1 atsakomybė gavus du išorinius signalus, juos padauginti iš koeficiento ir susumuoti. Testuojamas sumatoriaus įvykis, kai pasikeitė sumos operandas (funkcijos išvestinės reikšmė). Varža R* nekintama ir lygi 100. Koeficiento reikšmė $\beta_1 = 1$. Atliekamas veiksmas: Rezultatas = $X3 \cdot \beta_1 + X4 \cdot (-\beta_1)$.

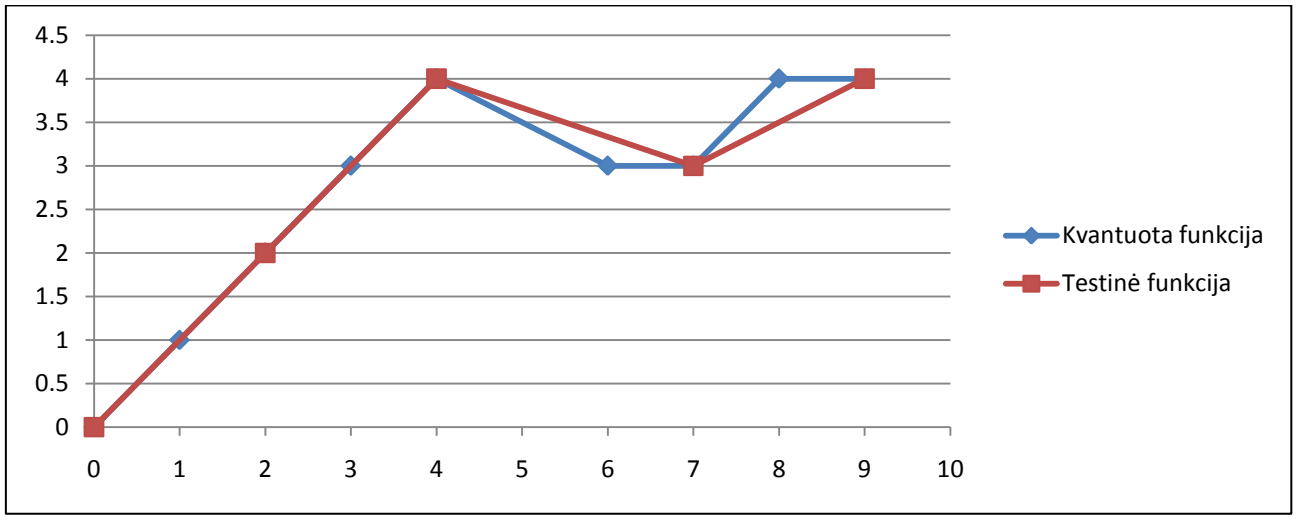
Lentelė 5 Sumatoriaus 4 testavimo rezultatai

X3	X4	Laukiamas rezultatas	Agregato rezultatas	Pastaba
0	0	0	0	Abi įvestys 0
5	0	-50	5	X1 > 0
0	5	50	-5	X2 > 0
5	5	0	0	Abi įvestys > 0
-5	0	50	-5	X1 < 0
0	-5	-50	5	X2 < 0
-5	-5	0	0	X1, X2 < 0

Pagal lentelės rezultatus matome, kad sumatorius veikia tvarkingai ir nedaro klaidų.

4.5.6. Integratoriaus testavimas

Siekiant patikrinti ar integratoriaus agregatas veikia korektiškai, buvo sudarytas testas. Pasirinktais laiko momentais integratoriui buvo perduodama nauja išvestinės reikšmė. Pasirinktas diskretizacijos lygis $\Delta Q = 1$. Gauti rezultatai (Pav. 14) rodo, kad integratorius tinkamai atlieka savo funkcijas.



Pav. 14 Testinės ir kvantuotos funkcijų grafikų palyginimas

4.6. Rezultatų analizė

Tiriama plyšinės jungties elgsena, kai plyšinė jungtis realizuota tiek kintančia, tiek nekintančia varža. Apibūdinti kaip keičiasi elektros grandinių elementais tekančios srovės grafikai ir kaip kintanti varža įtakoja šiuos procesus.

Tiriamos elektros grandinės elementų įtampos ir stipriai. Parinkti elektros grandinės elementų įverčiai:

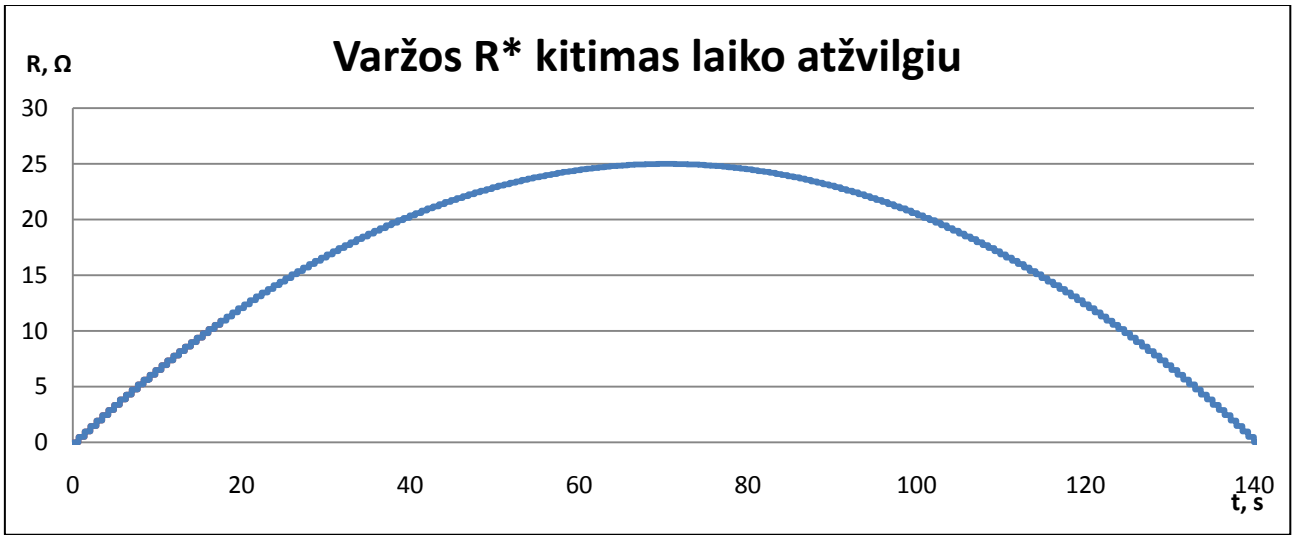
Lentelė 6 Elektros grandinės elementų parametrai

Elementas	Reikšmė
R_0	100 Ω
R_1	1000 Ω
R_2	1000 Ω
R*	Kintamos varžos atveju kinta nuo 0 Ω iki 25 Ω. Pastovios varžos atveju reikšmė 100 Ω.
C_1	0,001 F
C_2	0,001 F

Pirmiausia apžvelgiami elektros srovės procesai grandinėje, atskirai apžvelgiant, kai R* netiesinis ir tiesinis elementas (R* = const). Po to apibendrinami gauti rezultatai įvertinant kokią įtaką tiriamai elektros grandinei turi netiesinis varžos elementas.

Varža R* netiesinis elementas

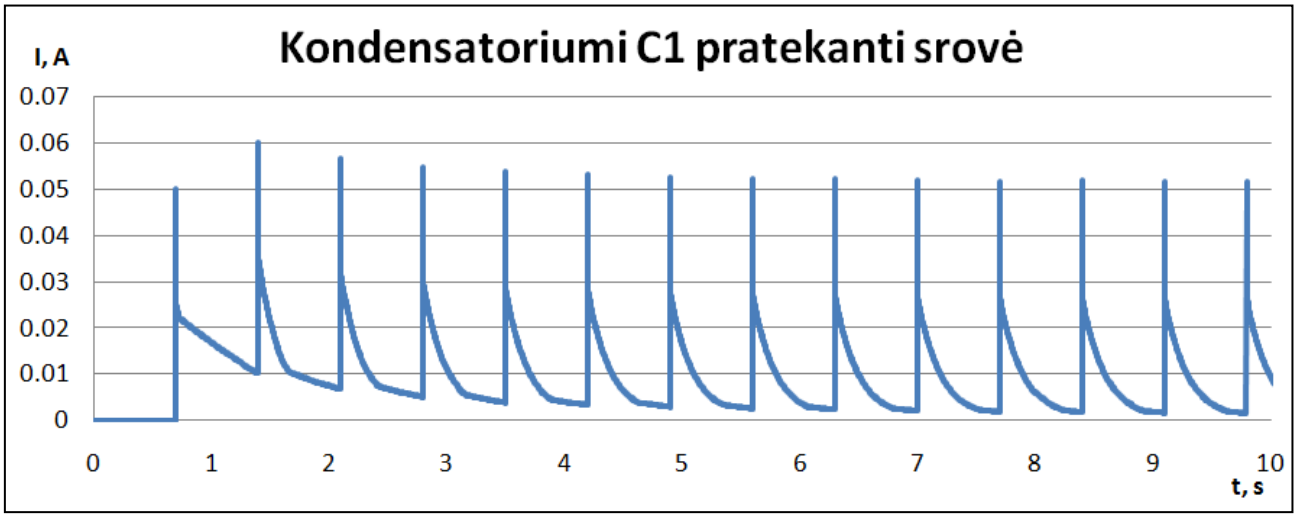
Elektros grandinės elementas R* kinta nuo 0 Ω iki 25 Ω. Imitacijos trukmė 140 sekundžių. Pav. 15 pavaizduotas varžos kitimas laiko atžvilgiu visos imitacijos trukmės atžvilgiu. Elemento R* reikšmė priklauso nuo išorinės įtampos. Naujas išorinis signalas (įtampa) paduodamas kas 0,7 sekundės.



Pav. 15 Varžos kitimas laike (išorinis signalas pasirodo kas 0,7 s, maksimali varžos reikšmė 25 Ω)

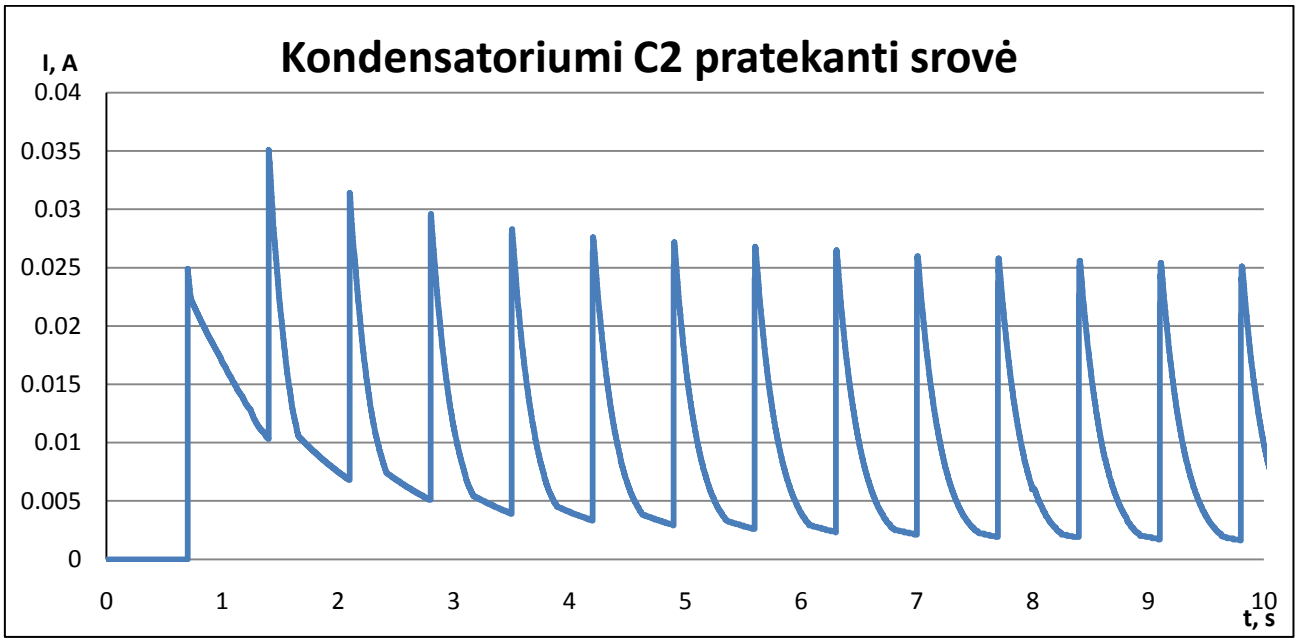
Kondensatoriumi C_1 tekanti elektros srovė pavaizduota Pav. 16 grafike. Dėl vaizdumo pateiktos pirmos 10 sekundžių. Nuo pirmojo grandinėje paduoto išorinio įtampos šuolio kondensatoriumi C_1 tekanti elektros srovė tuo momentu pakyla ir laikui einant iki sekančio įtampos

šuolio nusistovi. Kiekvienu nauju išoriniu įtampos šuoliu kondensatoriuje C_1 tęsiasi jau aprašytas procesas. Svarbu paminėti, kad pirmosiomis imitacijos sekundėmis kondensatoriumi C_1 elektros srovė nespėja nusistovėti ir sulaukiamas naujas įtampos šuolis, bet tolimesniu imitacijos metu tokia reakcija nestebima. Galima teigti, kad elektros srovei pradėjus tekėti grandine pirmosiomis sekundėmis vyksta aukščiau aprašyti procesai. Tai įtakoja kintančios varžos R* reikšmė, kuri pradiniu momentu yra 0.



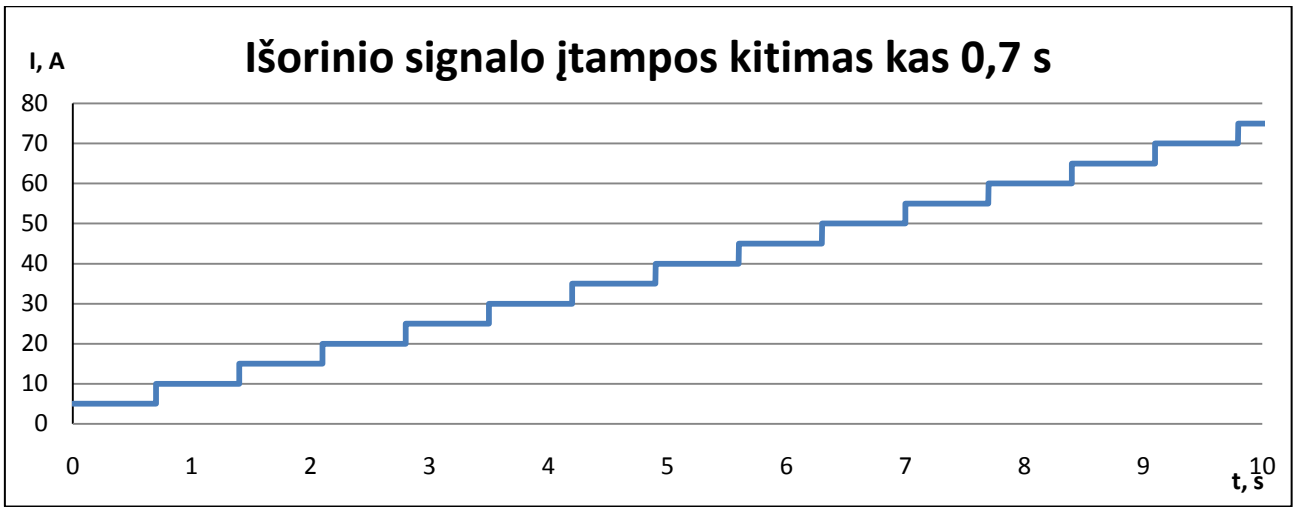
Pav. 16 Kondensatoriumi C_1 pratekanti srovė

Panašiai elgiasi ir kondensatoriumi C_2 pratekanti elektros srovė. Sulig kiekvienu išoriniu įtampos signalu stebimas momentinis srovės padidėjimas, kuris po to mažėja ir nusistovi.



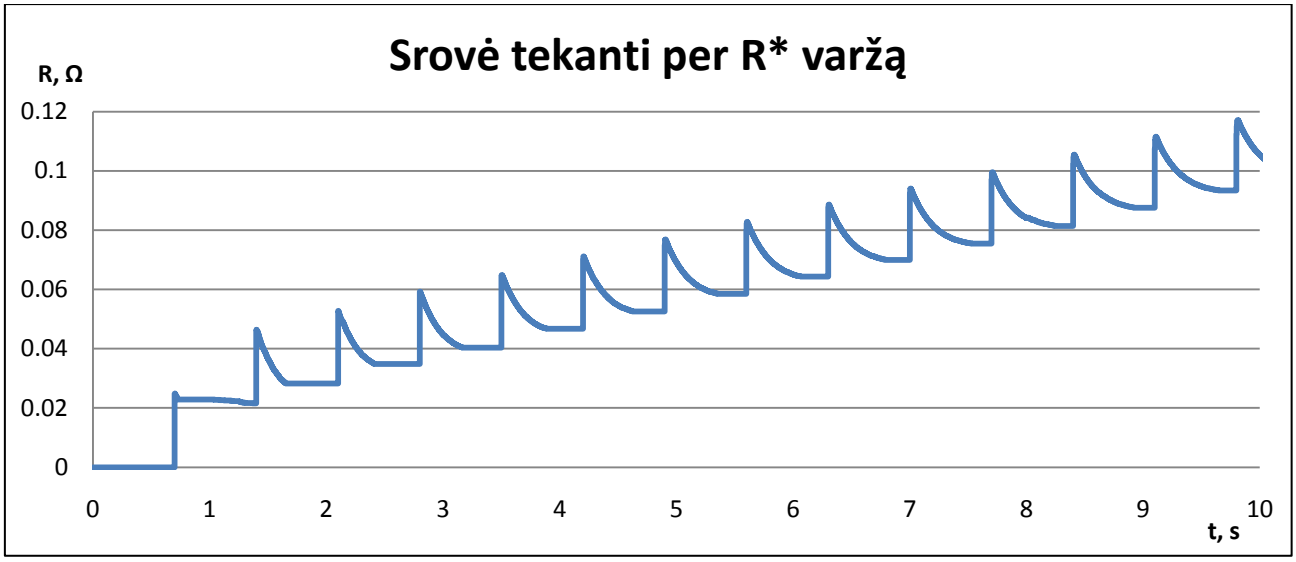
Pav. 17 Kondensatoriumi C_2 tekanti srovė

Kadangi išorinis įtampos signalas elektros gandine paduodamas kas 0,7 sekundės ir visada būna stipresnis 5 V nei prieš tai buvęs, tai grafikas turėtų būti laiptai. Būtent tokie rezultatai ir gaunami (žr. Pav. 18).



Pav. 18 Išorinio signalo (įtampos) kitimo grafikas

Elektros srovei tekant per kintamos varžos elementą (R*) stebimi panašūs procesai kaip ir kondensatoriuje, tik čia nusistovėjusi srovė yra didesnė už žingsniu ankščiau buvusią.

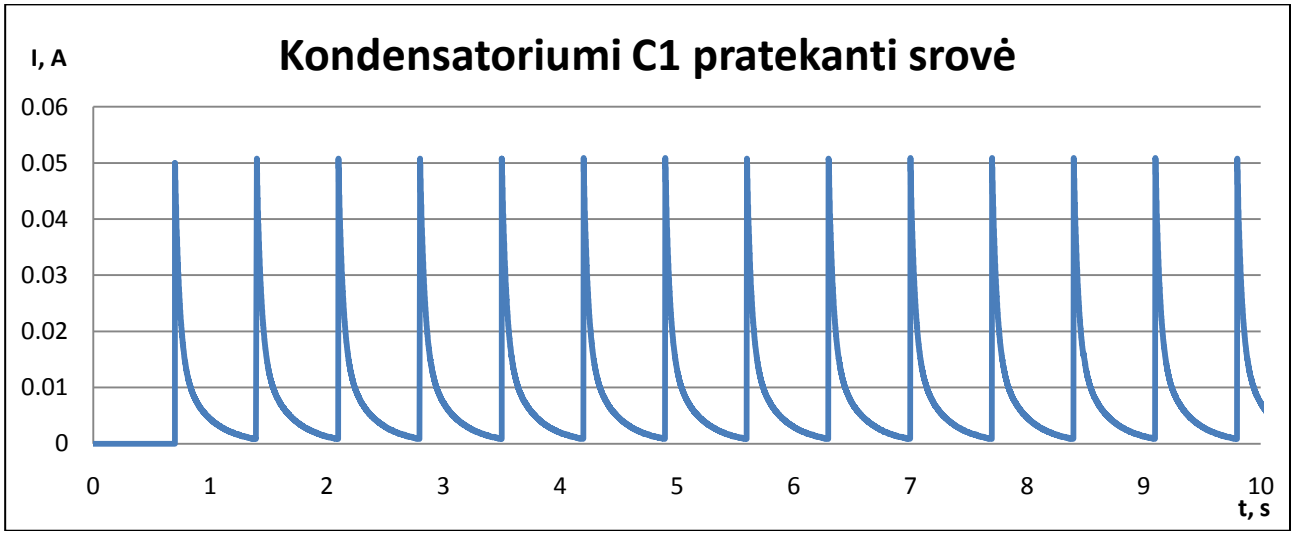


Pav. 19 Srovės, pratekančios per kintantį varžos R* elementą 10 sekundžių grafikas

Varža R* tiesinis elementas (R* = const)

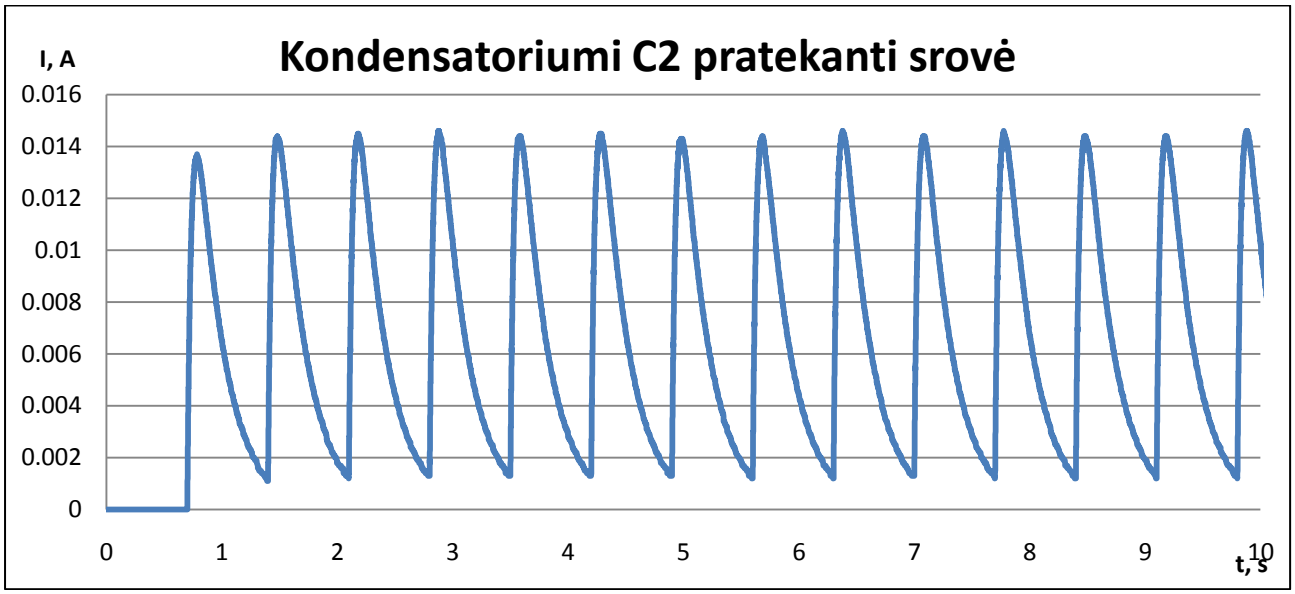
Aprašant sroves dėl vaizdumo grafikuose pavaizduoti duomenys bus pateikti 10 sekundžių laiko intervale.

Tiriant kondensatoriumi C_1 pratekančią srovę pastebime, kad ties kiekvienu išoriniu įtampos signalu, srovė momentiškai sustiprėja ir palaipsniui nusistovi (). Srovės reikšmės kiekviename žingsnyje pastovios, kai esant netiesiniam varžos R* elementui imitacijospradžioje stebime kiek kitokį procesą (žr. paveikslo Pav. 16 aprašymą).



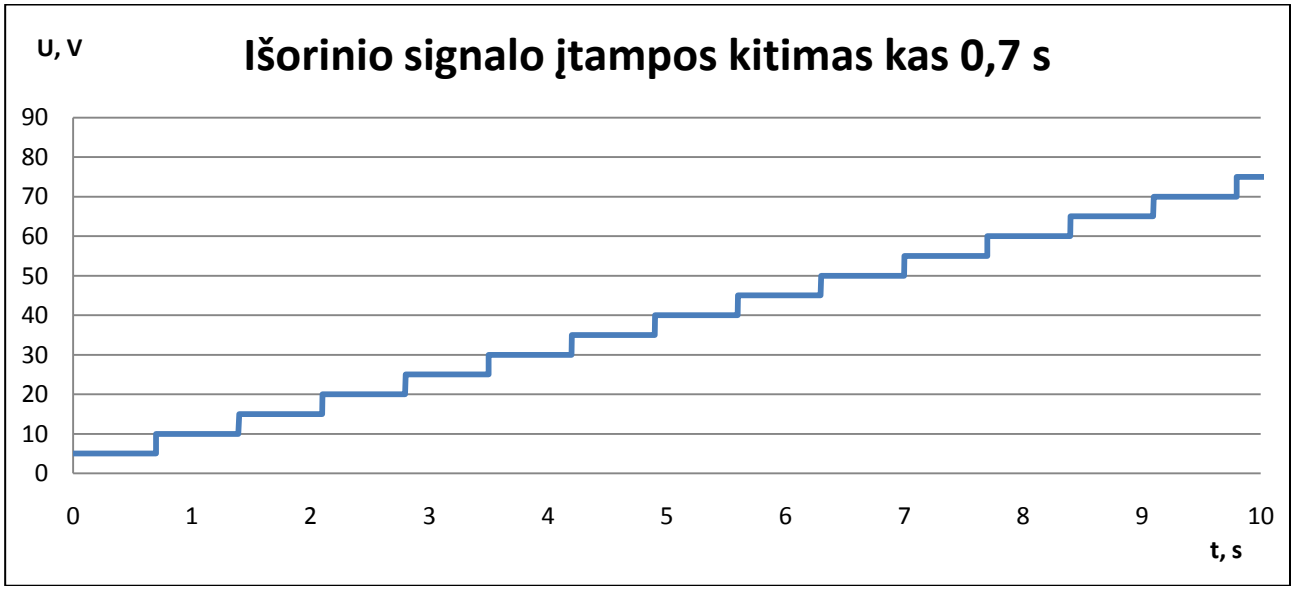
Pav. 20 Kondensatoriumi C_1 pratekanti srovė

Antrame kondensatoriuje elektros srovė elgiasi taip taip kaip ir pirmame. Ties išoriniais įtampos šuoliais stebimas įtampos padidėjimas ir nusistovėjimas. Jei laiko tarpas tarp išorinių įtampos šuolių būtų didesnis (pvz.: 2 sekundės), tai matytume kaip srovė tampa 0.



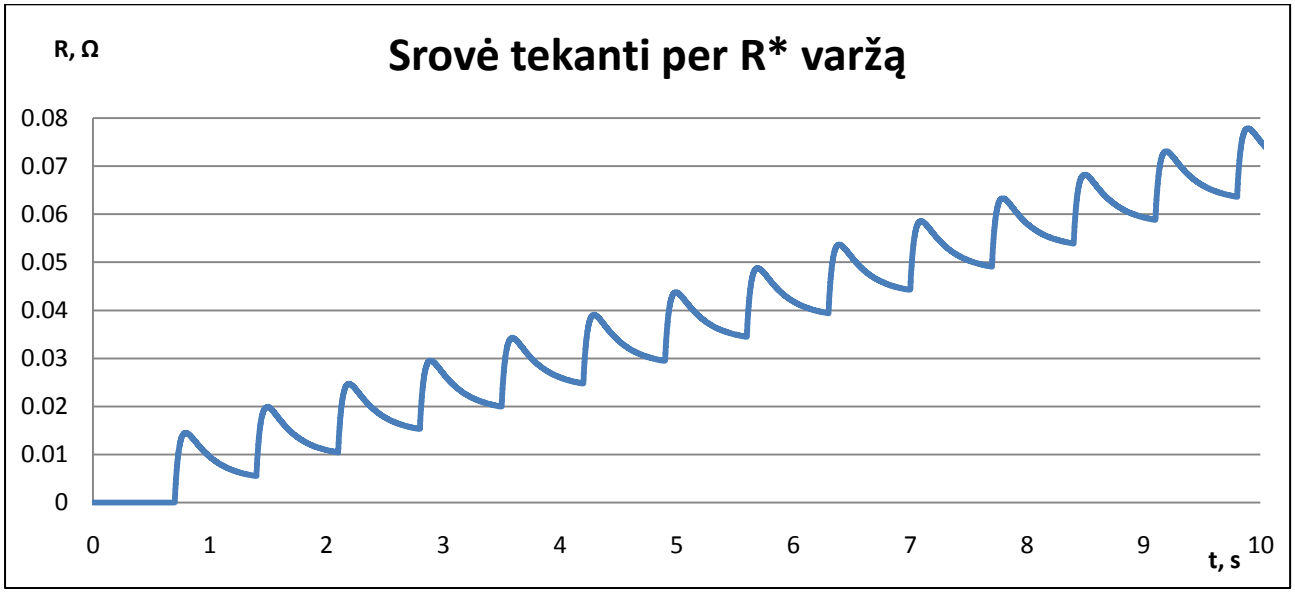
Pav. 21 Kondensatoriumi C_2 tekanti srovė

Išorinio signalo grafike kaip ir ankstesniame variante (kai R* netiesinis elementas) įtampos grafiko forma „laiptai“.



Pav. 22 Išorinio signalo (įtampos) kitimo grafikas

Elektros srovei tekant per kintamos varžos elementą (R*) stebimi panašūs procesai kaip ir kondensatoriuje, tik čia nusistovėjusi srovė yra didesnė už žingsniu ankščiau buvusią. Lyginant su grafiku, pavaizduotu esančių Pav. 19, kai kintama varžos elementas R* netiesinis, matome jog srovės nusistovėjimas staigesnis.



Pav. 23 Srovės, pratekančios per kintantį varžos R* elementą 10 sekundžių grafikas

5. IŠVADOS

- Pateiktas plyšinių jungčių agregatinio modeliavimo metodas tinkamas spręsti uždavinius, kuriuos galime aprašyti PLA specifikacija. Svarbu pažymėti, kad modelis privalo būti skaidomas į mažus agregatus. Panaudojus specialiai sukurtą programinę įrangą sėkmingai realizuotas tokio tipo agregatinis modelis. Remiantis šiuo metodu sėkmingai išspręstas plyšinės jungties imitacinis modelis su dviem ląstelėm. Aprašytas modelis yra tinkamas modeliuoti ilgesnius ląstelių tinklus. Šio metodo privalumas, kad tam tikri agregatai gali būti pritaikomi daugeliui modelių (su sąlyga jei panaudojamo agregato funkcionalumas atitinka reikiamą).
- Elektros grandine tekanti elektros srovė atitinka literatūroje aprašomus procesus, kuriais pasižymi plyšinė jungtis, todėl galime teigti, kad ištirtas imitacinis modelis veikia teisingai.
- Ištirta plyšinė jungtis, kai plyšinės jungties varža yra tiesinė, netiesinė. Gauti elektros srovės grafikai rodo, kad kintanti plyšinės jungties varža (varža priklauso nuo įtampos paduodamos elektros grandinei) įtakoja procesus vykstančius ląstelėse.

6. NAUDOTA LITERATŪRA

1) Paulauskas, N. et al. A Stochastic Four-State Model of Contingent Gating of Gap Junction Channels Containing Two “Fast” Gates Sensitive to Transjunctional Voltage. Biophysic Journal, 2009, volume 96, p. 3936-3948.
2) Paul D. L., Molecular cloning of cDNA for rat liver gap junction protein. 1986.
3) Harris A. L., Emerging issues of connexin channels: biophysics fills the gap. Q. Rev. Biophys. 34:325-427. 2001.
4) Sohl, G. and Willecke, K. Gap junctions and the connexin protein family. Cardiovasc. Res. 62(2), 2004:228-32.
5) Pranevičius, M. et al. Imitacinis tarpląstelinių plyšinių jungčių vartinio mechanizmo priklausomybės nuo įtampos modeliavimas. KTU, Verslo informatikos katedra. Kaunas, 2009. 5 p.
6) Spray, D. C., Harris, A. L. and Bennett, M. V. 1981. Gap junctional conductance is a simple and sensitive function of intracellular pH. Science. 211:712–715.
7) Bennett, M. V., and Verselis, V. K. 1992. Biophysics of gap junctions. Semin. Cell Biol. 3:29–47.
8) Bukauskas, F. F.; Verselis, V. K. Gap junction channel gating. Science Direct, 2004, p. 42-60.
9) Bukauskas FF, Angele AB, Verselis VK, Bennett MV. Coupling asymmetry of heterotypic connexin 45/ connexin 43-EGFP gap junctions: properties of fast and slow gating mechanisms. 2002.
10) Zak, R. 1973. Cell proliferation during cardiac growth. Am. J. Cardiol. 31:211–219 p.
11) MacCannell, K. A. et all. A Mathematical Model of Electrotonic Interactions between Ventricular Myocytes and Fibroblasts. Biophysical Journal, 2007, Volume 92, p. 4231-4132.
12) Miragoli, M., Gaudesius, G. and Rohr., S. 2006. Electrotonic modulation of cardiac impulse conduction by myofibroblasts. Circ. Res. 98: 801–810 p.
13) Jacquemet, V. Pacemaker activity resulting from the coupling with nonexcitable cells. Lausanne, 2006.
14) Kamkin, A., I. Kiseleva, and Isenberg, G.. 2003. Activation and inactivation of a non-selective cation conductance by local mechanical deformation of acutely isolated cardiac fibroblasts. Cardiovasc. Res. 57:793–803 p.
15) Kamkin, A., I. Kiseleva, Lozinsky, I. and Scholz., H. 2005. Electrical interaction of mechanosensitive fibroblasts and myocytes in the heart. Basic Res. Cardiol. 100:337–345 p.
16) Priebe, L., and Beuckelmann, D. J.. 1998. Simulation study of cellular electric properties in heart failure. Circ. Res. 82:1206–1223 p.

17) Umino, O., Maehara, M., Hidaka, S., Kita, S. and Hashimoto, Y. (1994) The network properties of bipolar-bipolar cell coupling in the retina of teleost fishes. Vis. Neurosci. 1 I, 533-548 p.
18) Poznanski, R. R.; Umino, O. Syncytial integration by a network of coupled bipolar cells in the retina. Progress in Neurobiology, 1997, Vol. 53, p. 273 to 291.
19) Spitzer, K. W. et al. Electrotonic modulation of electrical activity in rabbit atrioventricular node myocytes. Salt Lake City, Utah, 1997, p. H-767 to H-776.
20) Bukauskas, F. et al. Intercellular communication. p. 213-217.
21) Henriquez, A. P. et al. Influence of Dynamic Gap Junction Resistance on Impulse Propagation in Ventricular Myocardium: A Computer Simulation Study. Biophysical Journal, 2001, Volume 81, p. 2112-2121.
22) Dzemyda, G.; Kurasova, O.; Žilinskas, J. Daugiamačių duomenų vizualizavimo metodai. Vilnius, 2008. 2004 p.
23) Pranevičius, H. Sudėtingų sistemų formalizavimas ir analizė. Kaunas, 2008. 239 p.
24) Pranevičius, H., Simaitis, L. Piece-Linear Aggregates for Formal Specification and Simulation of Hybrid Systems: Pharmacokinetics Patient-Controlled Analgesia. 2011. 81 p.

7. PRIEDAS

LUA skriptų logikos funkcijos

sumator_S1.lua

```
function run_logic(X1, X2)
             file:write("\n Sumator1::run_logic(X1 = " .. X1 .. ", X2 = " .. X2 .. ")\n")
             file:write("Sumator1:: C1 = " .. C1 .. ", R0 = " .. R0 .. "\n")

             -- R0 - parametras (įvedama iš programos)
             beta1 = 1/(C1*R0)
             flux = X1*(-beta1) + X2*(beta1)
             file:write("Calculated values: flux = " .. flux .. "\n")

             flux = round(flux, 4)
             c_update_res(p_self, out_types[1], flux, out_names[1])
             file:write("Sumator1::run_logic() end\n")
             return "double",  flux
end
```

sumator_S2.lua

```
function run_logic(X1, X2, X3, X4, R_star)
             file:write("\nSumator2::run_logic( X1 = " .. X1 .. ", X2 = " .. X2 .. ", X3 = " .. X3
.. ", X4 = " .. X4 .. ")\n")
             file:write("Sumator2:: C1 = " .. C1 .. ", C2 = " .. C2 .. ", R1 = " .. R1 .. ", R_star
= " .. R_star .."\n")

             -- R1 - įvedama iš programos
             -- C1 - įvedama iš programos
             -- C2 - įvedama iš programos
             beta1 = (R1 - R_star)/(C1*R_star*R1)
             beta2 = 1/(C2*R_star)

             flux = X1*beta1 + X2*(-beta1) + X3*(-beta2) + X4*beta2
             flux = round(flux,4)
             file:write("Calculated value: flux = " .. flux .. "\n")

             c_update_res(p_self, out_types[1], flux, out_names[1])
             file:write("Sumator2::run_logic() end\n")
             return "double",  flux
end
```

sumator_S3.lua

```
function run_logic(X1, X2, X3, X4, R_star)
             file:write("\nSumator3::run_logic( X1 = " .. X1 .. ", X2 = " .. X2 .. ", X3 = " .. X3
.. ", X4 = " .. X4 .. ")\n")
             file:write("Sumator3:: C1 = " .. C1 .. ", C2 = " .. C2 .. ", R1 = " .. R1 .. ", R_star
= " .. R_star .."\n")

             -- C1 - įvedama iš programos
             -- C2 - įvedama iš programos
             beta1 = 1/(C1*R_star)
             beta2 = 1/(C2*R_star)
             flux = X1*beta1 + X2*(-beta1) + X3*(-beta2) + X4*beta2

             flux = round(flux,4)
             file:write("Calculated values: flux = " .. flux .. "\n")
             c_update_res(p_self, out_types[1], flux, out_names[1])
             file:write("Sumator3::run_logic() end\n")
             return "double",  flux
end
```

sumator_S4.lua

```
function run_logic(X3, X4)
             file:write("\nSumator4::run_logic( X3 = " .. X3 .. ", X4 = " .. X4 .. ")\n")
             file:write("Sumator4:: C1 = " .. C1 .. ", R2 = " .. R2 .. "\n")

             -- C2 - įvedama iš programos
             beta1 = 1/(C1*R2)
             flux = X3*(beta1) + X4*(-beta1)

             flux = round(flux,4)
             file:write("Calculated values: flux = " .. flux .. "\n")
             c_update_res(p_self, out_types[1], flux, out_names[1])
             file:write("Sumator4::run_logic() end\n")
```

```
            return "double",  flux
end
```

sumator_S5.lua

```
function run_logic(X1, u)
            file:write("\n Sumator5::run_logic(X1 = " .. X1 .. ", u = " .. u .. ")\n")
            flux = X1 + u
            file:write("Calculated values: flux plus output signal = " .. flux .. "\n")

            flux = round(flux, 4)
            c_update_res(p_self, out_types[1], flux, out_names[1])
            file:write("Sumator5::run_logic() end\n")
            return "double",  flux
end
```

integrator.lua

```
--req_type 0 == isorinis ivykis
--req_type 1 == pilnai isorinis 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           pilnai isorinis ivykis (flux == 0)
--8   pilnai isorinis ivykis (flux > 0)
--9           pilnai isorinis 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("  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
```

main_logic.lua

```
function run_logic()
      file:write("file_logic::run_logic() \n")

      t = 0   -- sisteminis imitacijos laikas
      counter = 0          -- skaitliukas isoriniams ivykiams fiksuoti
      req_t = 0
      pred1 = math.huge
      pred2 = math.huge
      pred3 = math.huge
      pred4 = math.huge

      -- Isorinio signalo momentai
      file:write("Generating external moments..\n")
      t_max = 200
      delta_t = 0.7
      --temp = math.floor((t_max+t_max)/delta_t) -- kiek reikes sugeneruoti reiksmiu kas delta_t
laiko momentu
      external_time = {}
      in_du = {}
      external_time[0] = 0
      in_du[0] = 0
      j = 1
      while t <= t_max do
            t = t + delta_t
            external_time[j] = t
            in_du[j], ok = c_run_logic(graph_pointers[1]["pointer"], 1,"double", j)
```

```
            file:write("j = "..j.."; time = "..external_time[j].."; Value = "..in_du[j].."\n")
            j = j + 1
      end
      t = 0
      file:write("External moments generated.\n")

      req_t = external_time[counter]

      k = 0
      -- imitacijos trukme 50 sekundziu
      while t <= t_max and k < 35000 do
            k = k + 1

            min = find_min(req_t, pred1, pred2, pred3, pred4)
            R_star = 100
            --R_star = 25 - (u - 5) * (u - 5)

            file:write("R_star = " .. R_star .. "/n")
            if min == 1 then
                  file:write("---------------------------\n")
                  file:write("Isorinis ivykis t = "..t.." min = "..min.."\n")
                  t = req_t

                  u = in_du[counter]
                  if t == pred1 then
                        sigma_i1, flux1, Q_level_i1, err_i1, ok = c_run_logic(integr1, 4, "double",
t, "int", 2, "double", flux1)
                  end
            if t == pred2 then
                  sigma_i2, flux2, Q_level_i2, err_i2, ok = c_run_logic(integr2, 4, "double", t,
"int", 2, "double", flux2)
            end
            if t == pred3 then
                  sigma_i3, flux3, Q_level_i3, err_i3, ok = c_run_logic(integr3, 4, "double", t,
"int", 2, "double", flux3)
            end
            if t == pred4 then
                  sigma_i4, flux4, Q_level_i4, err_i4, ok = c_run_logic(integr4, 4, "double", t,
"int", 2, "double", flux4)
            end

              -- sumator5
              ext_flux, ok = c_run_logic(sum5, 1, "double", Q_level_i1, "double", u, "int", 1)
              -- sumator1
              flux1, ok = c_run_logic(sum1, 1, "double", ext_flux, "double", Q_level_i2)
              -- sumator2
              flux2, ok = c_run_logic(sum2, 1, "double", ext_flux, "double", Q_level_i2, "double",
Q_level_i3, "double", Q_level_i4, "double", R_star)
              -- sumator3
              flux3, ok = c_run_logic(sum3, 1, "double", ext_flux, "double", Q_level_i2, "double",
Q_level_i3, "double", Q_level_i4, "double", R_star)

              -- integrator 1
              sigma_i1, flux1, Q_level_i1, err_i1, ok = c_run_logic(integr1, 4, "double", t, "int",
0, "double", flux1)
              -- integrator 2
              sigma_i2, flux2, Q_level_i2, err_i2, ok = c_run_logic(integr2, 4, "double", t, "int",
0, "double", flux2)
              -- integrator 3
              sigma_i3, flux3, Q_level_i3, err_i3, ok = c_run_logic(integr3, 4, "double", t, "int",
0, "double", flux3)

              sigma_i1 = round(sigma_i1,4)
              sigma_i2 = round(sigma_i2,4)
              sigma_i3 = round(sigma_i3,4)

              if flux1 == 0 then pred1 = math.huge else pred1 = t + sigma_i1      end
              if flux2 == 0 then pred2 = math.huge else pred2 = t + sigma_i2      end
              if flux3 == 0 then pred3 = math.huge else pred3 = t + sigma_i3 end
              if flux4 == 0 then pred4 = math.huge end

              file:write("---------------------------\n")
              counter = counter + 1
              req_t = external_time[counter]
              elseif min == 2 then
                  t = pred1
                  file:write("t = "..t.." min = "..min.."\n")

                  if t == pred1 then
```

```
                        sigma_i1, flux1, Q_level_i1, err_i1, ok = c_run_logic(integr1, 4,
"double", t, "int", 2, "double", flux1)
                end
                if t == pred2 then
                        sigma_i2, flux2, Q_level_i2, err_i2, ok = c_run_logic(integr2, 4,
"double", t, "int", 2, "double", flux2)
                end
                if t == pred3 then
                        sigma_i3, flux3, Q_level_i3, err_i3, ok = c_run_logic(integr3, 4,
"double", t, "int", 2, "double", flux3)
                end
                if t == pred4 then
                        sigma_i4, flux4, Q_level_i4, err_i4, ok = c_run_logic(integr4, 4,
"double", t, "int", 2, "double", flux4)
                end

                -- sumator5
                ext_flux, ok = c_run_logic(sum5, 1, "double", Q_level_i1, "double", u, "int", 1)
                -- sumator1
                flux1, ok = c_run_logic(sum1, 1, "double", ext_flux, "double", Q_level_i2)
                -- sumator2
                flux2, ok = c_run_logic(sum2, 1, "double", ext_flux, "double", Q_level_i2,
"double", Q_level_i3, "double", Q_level_i4, "double", R_star)
                -- sumator3
                flux3, ok = c_run_logic(sum3, 1, "double", ext_flux, "double", Q_level_i2,
"double", Q_level_i3, "double", Q_level_i4, "double", R_star)

                -- integrator 1
                sigma_i1, flux1, Q_level_i1, err_i1, ok = c_run_logic(integr1, 4, "double", t,
"int", 0, "double", flux1)
                -- integrator 2
                sigma_i2, flux2, Q_level_i2, err_i2, ok = c_run_logic(integr2, 4, "double", t,
"int", 0, "double", flux2)
                -- integrator 3
                sigma_i3, flux3, Q_level_i3, err_i3, ok = c_run_logic(integr3, 4, "double", t,
"int", 0, "double", flux3)

                sigma_i1 = round(sigma_i1,4)
                sigma_i2 = round(sigma_i2,4)
                sigma_i3 = round(sigma_i3,4)

                if flux1 == 0 then pred1 = math.huge else pred1 = t + sigma_i1  end
                if flux2 == 0 then pred2 = math.huge else pred2 = t + sigma_i2  end
                if flux3 == 0 then pred3 = math.huge else pred3 = t + sigma_i3 end
                if flux4 == 0 then pred4 = math.huge end
            elseif min == 3 then
                t = pred2
                file:write("t = "..t.." min = "..min.."\n")

                if t == pred1 then
                        sigma_i1, flux1, Q_level_i1, err_i1, ok = c_run_logic(integr1, 4,
"double", t, "int", 2, "double", flux1)
                        file:write("internal1\n")
                end
                if t == pred2 then
                        sigma_i2, flux2, Q_level_i2, err_i2, ok = c_run_logic(integr2, 4,
"double", t, "int", 2, "double", flux2)
                        file:write("internal2\n")
                end
                if t == pred3 then
                        sigma_i3, flux3, Q_level_i3, err_i3, ok = c_run_logic(integr3, 4,
"double", t, "int", 2, "double", flux3)
                        file:write("internal3\n")
                end
                if t == pred4 then
                        sigma_i4, flux4, Q_level_i4, err_i4, ok = c_run_logic(integr4, 4,
"double", t, "int", 2, "double", flux4)
                end

                -- sumator1
                flux1, ok = c_run_logic(sum1, 1, "double", ext_flux, "double", Q_level_i2)
                -- sumator2
                flux2, ok = c_run_logic(sum2, 1, "double", ext_flux, "double", Q_level_i2,
"double", Q_level_i3, "double", Q_level_i4, "double", R_star)
                -- sumator3
                flux3, ok = c_run_logic(sum3, 1, "double", ext_flux, "double", Q_level_i2,
"double", Q_level_i3, "double", Q_level_i4, "double", R_star)

                -- integrator 1
```

```
                sigma_i1, flux1, Q_level_i1, err_i1, ok = c_run_logic(integr1, 4, "double", t,
"int", 0, "double", flux1)
                -- integrator 2
                sigma_i2, flux2, Q_level_i2, err_i2, ok = c_run_logic(integr2, 4, "double", t,
"int", 0, "double", flux2)
                -- integrator 3
                sigma_i3, flux3, Q_level_i3, err_i3, ok = c_run_logic(integr3, 4, "double", t,
"int", 0, "double", flux3)

                sigma_i1 = round(sigma_i1,4)
                sigma_i2 = round(sigma_i2,4)
                sigma_i3 = round(sigma_i3,4)

                if flux1 == 0 then pred1 = math.huge else pred1 = t + sigma_i1  end
                if flux2 == 0 then pred2 = math.huge else pred2 = t + sigma_i2  end
                if flux3 == 0 then pred3 = math.huge else pred3 = t + sigma_i3 end
                if flux4 == 0 then pred4 = math.huge end
          elseif min == 4 then
                t = pred3
                file:write("t = "..t.." min = "..min.."\n")

                if t == pred1 then
                        sigma_i1, flux1, Q_level_i1, err_i1, ok = c_run_logic(integr1, 4,
"double", t, "int", 2, "double", flux1)
                end
                if t == pred2 then
                        sigma_i2, flux2, Q_level_i2, err_i2, ok = c_run_logic(integr2, 4,
"double", t, "int", 2, "double", flux2)
                end
                if t == pred3 then
                        sigma_i3, flux3, Q_level_i3, err_i3, ok = c_run_logic(integr3, 4,
"double", t, "int", 2, "double", flux3)
                end
                if t == pred4 then
                        sigma_i4, flux4, Q_level_i4, err_i4, ok = c_run_logic(integr4, 4,
"double", t, "int", 2, "double", flux4)
                end

                -- sumator2
                flux2, ok = c_run_logic(sum2, 1, "double", ext_flux, "double", Q_level_i2,
"double", Q_level_i3, "double", Q_level_i4, "double", R_star)
                -- sumator3
                flux3, ok = c_run_logic(sum3, 1, "double", ext_flux, "double", Q_level_i2,
"double", Q_level_i3, "double", Q_level_i4, "double", R_star)
                -- sumator4
                flux4, ok = c_run_logic(sum4, 1, "double", Q_level_i3, "double", Q_level_i4)

                -- integrator 2
                sigma_i2, flux2, Q_level_i2, err_i2, ok = c_run_logic(integr2, 4, "double", t,
"int", 0, "double", flux2)
                -- integrator 3
                sigma_i3, flux3, Q_level_i3, err_i3, ok = c_run_logic(integr3, 4, "double", t,
"int", 0, "double", flux3)
                -- integrator 4
                sigma_i4, flux4, Q_level_i4, err_i4, ok = c_run_logic(integr4, 4, "double", t,
"int", 0, "double", flux4)

                sigma_i2 = round(sigma_i2,4)
                sigma_i3 = round(sigma_i3,4)
                sigma_i4 = round(sigma_i4,4)

                if flux1 == 0 then pred1 = math.huge end
                if flux2 == 0 then pred2 = math.huge else pred2 = t + sigma_i2  end
                if flux3 == 0 then pred3 = math.huge else pred3 = t + sigma_i3 end
                if flux4 == 0 then pred4 = math.huge else pred4 = t + sigma_i4  end
             elseif min == 5 then
                t = pred4
                file:write("t = "..t.." min = "..min.."\n")

                if t == pred1 then
                        sigma_i1, flux1, Q_level_i1, err_i1, ok = c_run_logic(integr1, 4,
"double", t, "int", 2, "double", flux1)
                end
                if t == pred2 then
                        sigma_i2, flux2, Q_level_i2, err_i2, ok = c_run_logic(integr2, 4,
"double", t, "int", 2, "double", flux2)
                end
                if t == pred3 then
                        sigma_i3, flux3, Q_level_i3, err_i3, ok = c_run_logic(integr3, 4,
"double", t, "int", 2, "double", flux3)
```

```
                end
                if t == pred4 then
                        sigma_i4, flux4, Q_level_i4, err_i4, ok = c_run_logic(integr4, 4,
"double", t, "int", 2, "double", flux4)
                end

                -- sumator2
                flux2, ok = c_run_logic(sum2, 1, "double", ext_flux, "double", Q_level_i2,
"double", Q_level_i3, "double", Q_level_i4, "double", R_star)
                -- sumator3
                flux3, ok = c_run_logic(sum3, 1, "double", ext_flux, "double", Q_level_i2,
"double", Q_level_i3, "double", Q_level_i4, "double", R_star)
                -- sumator4
                flux4, ok = c_run_logic(sum4, 1, "double", Q_level_i3, "double", Q_level_i4)

                -- integrator 2
                sigma_i2, flux2, Q_level_i2, err_i2, ok = c_run_logic(integr2, 4, "double", t,
"int", 0, "double", flux2)
                -- integrator 3
                sigma_i3, flux3, Q_level_i3, err_i3, ok = c_run_logic(integr3, 4, "double", t,
"int", 0, "double", flux3)
                -- integrator 4
                sigma_i4, flux4, Q_level_i4, err_i4, ok = c_run_logic(integr4, 4, "double", t,
"int", 0, "double", flux4)

                sigma_i2 = round(sigma_i2,4)
                sigma_i3 = round(sigma_i3,4)
                sigma_i4 = round(sigma_i4,4)

                if flux1 == 0 then pred1 = math.huge end
                if flux2 == 0 then pred2 = math.huge else pred2 = t + sigma_i2  end
                if flux3 == 0 then pred3 = math.huge else pred3 = t + sigma_i3 end
                if flux4 == 0 then pred4 = math.huge else pred4 = t + sigma_i4  end
            end
            c_update_res(p_self, out_types[1], ext_flux, out_names[1])
            c_update_res(p_self, out_types[2], Q_level_i2, out_names[2])
            c_update_res(p_self, out_types[3], Q_level_i3, out_names[3])
            c_update_res(p_self, out_types[4], Q_level_i4, out_names[4])
            c_update_res(p_self, out_types[5], flux1, out_names[5])
            c_update_res(p_self, out_types[6], flux2, out_names[6])
            c_update_res(p_self, out_types[7], flux3, out_names[7])
            c_update_res(p_self, out_types[8], flux4, out_names[8])
            c_update_res(p_self, out_types[9], in_du[counter], out_names[9])
            c_update_res(p_self, out_types[10], t, out_names[10])
            c_update_res(p_self, out_types[11], R_star, out_names[11])
      end
end
```